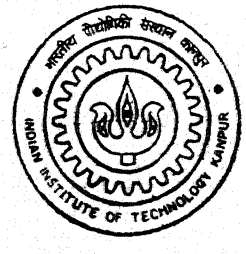
Obstacle Mapping and Inverse Kinematics of Hyper-Redundant Manipulators

by

AMIT TANDON

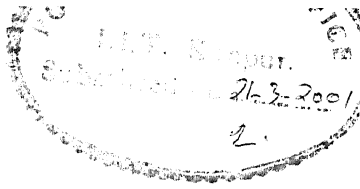


DEPARTMENT OF MECHANICAL ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY, KANPUR

March, 2001

CERTIFICATE



This is to certify that the work under the thesis titled **Obstacle Mapping and Inverse Kinematics of Hyper-redundant Manipulators** by **Amit Tandon (Roll No. 9910502)** has been carried out under my supervision and this work has not been submitted elsewhere for a degree.

Bhaskar Dasgupta
21/3/2001

Dr. Bhaskar Dasgupta
Assistant Professor
Department of Mechanical Engineering
IIT Kanpur

Dedicated to

Mom, Dad and Sis

Acknowledgement

I express my sincere thanks to my thesis supervisor Dr. Bhaskar Dasgupta for his constant encouragement, excellent guidance and invaluable suggestions. His seemingly obvious solutions to the most daunting of my problems strengthened my resolve in the dictum that nothing is impossible. He was always available at times of my need and has been quite understanding throughout.

I am greatly indebted to all my teachers for their endeavour to make me learn as much as possible. I am particularly thankful to Dr. K. Deb and Tushar for helping me with the application of GA and also providing me the code for a simple GA.

I am thankful to all my batchmates for providing the right atmosphere to hone my skills. I am particularly grateful to Jaydeep and Parthasarathy for their constant support and encouragement. I must also thank Sanjay and Pavan for helping me out when required and Ankur, Shamik and Kaushik for their nice company.

Last but not the least, I take this opportunity to thank everyone for making my stay at IIT Kanpur such a pleasing one.

Amit Tandon

Contents

List of Figures

Abstract

A live problem in modern day robotics is to construct highly dextrous manipulators to be used for maintenance purposes. Hyper-redundant manipulators provide one solution to this problem.

Manipulators in which the actuated degrees of freedom exceeds the minimal number required to perform a particular task are termed as 'redundant manipulators'. Hyper-redundant manipulators are redundant manipulators with a large degree of redundancy. Because of their highly articulated structures, these robots are well suited for operation in highly constrained environments. Such high redundancy also helps in obstacle avoidance, singularity avoidance, workspace enhancement and performance optimization.

For designing a hyper-redundant manipulator, an important step is to plan its path. Since hyper-redundant manipulators have a large number of degrees of freedom, Configuration space (C-space) based path planning is a promising prospect.

For C-space based path planning, an exhaustive description of the C-space is a pre-requisite. In addition to this, to find a feasible path, all candidate solutions are required to be known. The aim of this work is to provide the above mentioned inputs for path planning and synthesis. In this work, a way to map obstacles from Cartesian space to C-space is presented. An attempt is also made to represent different C-obstacles separately. Finally, a way to compute the inverse kinematics solutions is presented. The treatment is done for both 2-D and 3-D cases.

Chapter 1

Introduction

1.1 Hyper-Redundant Manipulators

The word *redundant* is used in the context of robotic manipulators to indicate that the number of actuated degrees of freedom exceeds the minimal number required to perform a particular task. For instance, a manipulator required to position and orient an object in space needs six actuated degrees of freedom and so a manipulator with seven or more degrees of freedom is redundant with respect to this task. "Hyper-Redundant" manipulators are redundant manipulators with a very large degree of redundancy. These manipulators are analogous in morphology and operation to 'snakes', 'elephant trunks' or 'tentacles'. Because of their highly articulated structures, these robots are well suited for operation in highly constrained environments and can be designed to have greater robustness with respect to mechanical failure than manipulators with a low degree of redundancy. Such high redundancy also helps in obstacle avoidance, singularity avoidance, workspace enhancement, performance optimization and improvement in reliability leading to safer operational situations.

To date, hyper-redundant manipulators have remained largely a laboratory curiosity. There are a number of reasons for this.

- standard kinematic techniques have not been particularly efficient or well-suited to the needs of hyper-redundant robot task modeling.
- the mechanical design and implementation of hyper-redundant robots

has been perceived as unnecessarily complex.

1.2 Problem of Path Planning

Many times in the field of robotics we come across situations where we have a manipulator fixed inside a workcell. There are a number of obstacles of various shapes and sizes located at different places inside the workcell and with different orientations. The problem of path planning is that we are given a start point and a goal point; the end effector of the robot has to be moved in such a way that while moving from the start point to the goal point it should not strike any obstacle or any workcell wall or any link of itself.

Thus we come to the important requisites of a good path, which are

- It should avoid obstacles.
- It should avoid workcell boundaries.
- It should avoid self-intersection of the manipulator.

A path that guarantees all the above three can be taken to be a feasible path.

Now, there are numerous ways to tackle this path planning problem. One way is to do the path planning in Cartesian space itself. But for hyper-redundant manipulators which have a large number of degrees of freedom, it is quite cumbersome to do it in Cartesian space. Another way to attack this problem is to do the planning in Configuration space (C-space). In this approach, the obstacles and the workcell boundaries are mapped from the Cartesian space to the Configuration space and a path is searched avoiding the C-space obstacles.

1.3 Inverse Kinematics Problem

Given the position and orientation of the end-effector of a manipulator, the inverse kinematics problem is the search for a suitable set of joint variables that cause the end-effector to take that particular position and orientation in

the workspace. For a hyper-redundant manipulator having a large number of degrees of freedom the number of solutions is infinite. Since the number of joint variables is quite large in this case, analytical methods prove to be inadequate. Optimization techniques, especially genetic algorithms (GAs), provide one way of finding a large number of these solutions.

1.4 Literature Review

Research strictly addressing the problem of path planning in C-space is fairly recent, although this idea of planning the path in C-space was introduced by Lozano-Peréz [1] way back in 1983. After that, there has been some research in this field, but most efforts get bogged down by the complexity of the problem. First of all, it is considered very difficult to generate the C-space obstacles precisely. Even when there is a precise description of the C-space, most researchers try to find a path by searching in the n-dimensional C-space. This exercise becomes quite complex for hyper-redundant manipulators. In fact, because of its complexity, not much work has been reported on C-space based path planning of hyper-redundant manipulators.

Most of the first planners were based on extensive searches in the configuration space (R A Brooks and T Lozano-Peréz [2]). When used in spaces with more than three dimensions, these approaches showed that it is practically impossible to explore all the space. Potential field techniques are thought of for on-line use (Khatib [3]), but they have one important drawback: the generation of local minima. Barraquand et al [4] describe numerical potential fields free of local minima. However, these potentials require an exploration of the entire configuration space. This method has been successfully used in robots with less than four degrees of freedom.

Ma and Kyono [5] provide an obstacle avoidance scheme for planar hyper-redundant manipulators. It is based on analysis in the defined posture space where three parameters are used to determine the hyper-redundant manipulator configuration. Another work by Kavraki and Latombe [6] proposes a pre-processing stage and a planning stage. Pre-processing generates a network of collision free nodes. Planning then connects any given initial and

final configurations of the robot to two nodes of the network and computes a path through the network between these two nodes. Lengyel et al [7] introduce a planner that uses standard graphics hardware to rasterize C-space obstacles into a series of bitmap slices and then uses dynamic programming to create a navigation function and to calculate paths in this rasterized space.

Ilari and Torras [8] describe a heuristic approach to 2-D path planning for a rigid mobile body in C-space. Bajaj and Kim [9] present an algebraic algorithm to generate C-space obstacle boundaries where the obstacles are given by patches of algebraic surfaces.

Considerable work has been done in inverse kinematics solutions for manipulators in general but not much on hyper-redundant manipulators in particular. A closed-form solution of the inverse kinematics of a redundant manipulator is in general not available. One method proposed by Gravagne and Walker [10] approaches inverse kinematics for a planar hyper-redundant manipulator by decomposition into either a natural or a wavelet basis. Xu and Nechyba [11] solve the inverse kinematics problem using fuzzy logic.

1.5 Scope of Work

This work is divided into two parts. The first part of the work is concerned with obstacle mapping. The obstacles comprise of:

- Physical obstacles present inside the workcell.
- Workcell boundaries.
- Self-intersection of the manipulator.

These obstacles are mapped from Cartesian space to Configuration space. The scene after the mapping is that we have a host of points in the C-space of the manipulator; some are obstacle points, some are boundary points, some are self-intersection points and the rest are 'free' points. Free points represent those configurations of the manipulator where it does not hit any kind of obstacle. A feasible path has to pass through these free points. An attempt is also made to get separate non-overlapping C-obstacles.

The second part of the work deals with performing inverse kinematics of the hyper-redundant manipulator using GA. Since the number of solutions are infinite, the function space is highly multimodal. Here we use the concept of 'niching and speciation' to zero-in on solutions lying on maximum number of peaks. This gives us 'solution clusters' which reflect a wide variety of inverse kinematic solutions.

1.5.1 Importance

The current work is the first step of a larger project, which is to realize and develop a robotic system by using redundancy to work in an environment with target locations difficult to access. A large number of tasks in the routine inspection and maintenance of plants consist of quite simple tasks that can be performed by robots.. This when compounded with a situation of the plant being inhospitable for human beings due to high temperature, toxic or radioactive substances makes an automated or robotic maintenance/inspection imperative. However the situation gets complicated as many of the inspection sites in the plant are in locations that are difficult to access manually as well as by a robot. In such a scenario, the need is to devise a manipulator with a large number of degrees of freedom so that the inspection sites can be accessed in a serpentine fashion rather than in a conventional anthropomorphic manner. This calls for a hyper-redundant manipulator, which indeed comes with its own theoretical and practical challenges. In the present work, the analysis of such 'complicated' workcells is addressed, the results of which will be vital for the next phases, namely path planning and design.

1.6 Assumptions

- The manipulator comprises of links connected in series through revolute or prismatic joints.
- The workcell considered in the problem is in the form of a rectangular parallelopiped.

- The links of the manipulator are circular in cross section and taper along their length.
- Obstacles comprise of simple primitives like parallelopipeds and cylinders.

Except for the first one, all assumptions have been made for the current implementation only and, in general, do not affect the problem formulation.

1.7 Organization of the Thesis

- Chapter 2 presents the way in which the workcell, physical obstacles and the manipulator are modeled for this work.
- Chapter 3 describes the complete algorithm used for performing obstacle mapping and getting C-obstacles.
- Chapter 4 provides an insight into the domain of genetic algorithms and describes how to use them to get inverse kinematics solutions.
- Chapter 5 gives the various kinds of results obtained for obstacle mapping and inverse kinematics.
- Chapter 6 summarizes the whole work and proposes its future scope.
- Appendices A and B summarize some well-known background material for easy reference, while appendices C and D are meant to form a handy reference for the program developed.

Chapter 2

Workcell and Manipulator Modeling

The first step to proceed towards obstacle mapping is to model the workcell and the manipulator in a convenient manner.

2.1 Workcell Modeling

The workcell consists of the room and the physical obstacles which are located inside it.

2.1.1 Room

The room we have in hand is in the shape of a rectangular parallelopiped (Fig 2.1). The three dimensions: length, breadth and height need to be specified to have its complete description. A Universal frame ({U} frame) is attached to its far-bottom-left corner. All locations inside the room are described with respect to this {U} frame.

2.1.2 Obstacles

The obstacles are modeled as physical objects which are made up of simple primitives. A frame is attached to each obstacle. The position and orientation of the obstacle inside the room is given by specifying the position and orientation of this frame with respect to the {U} frame.

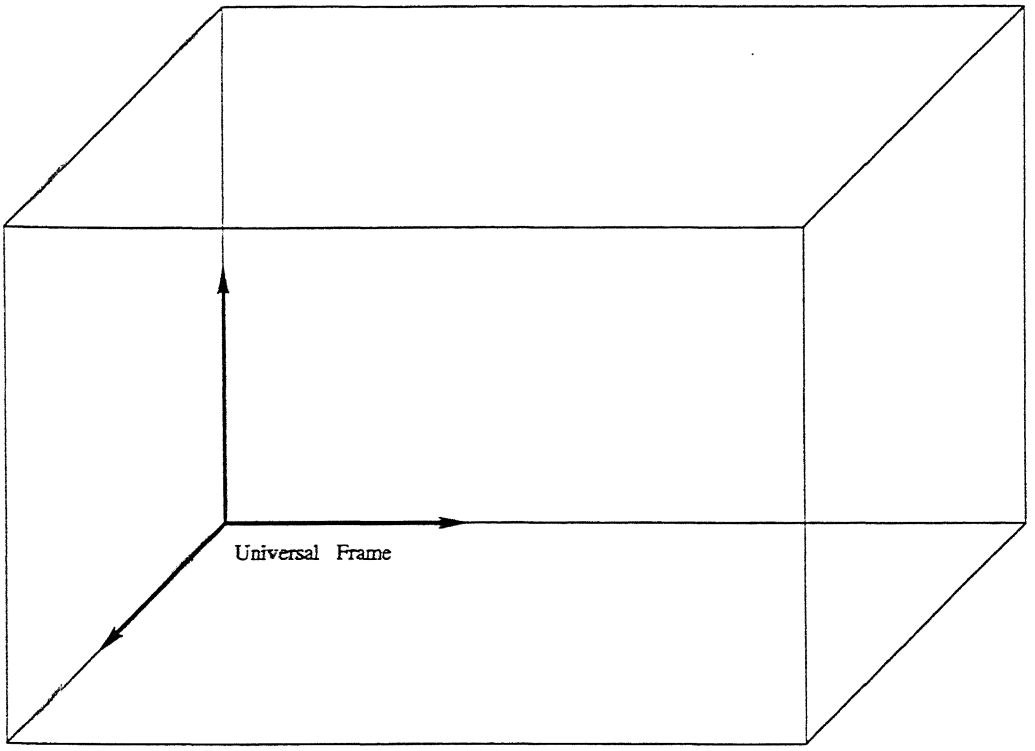


Figure 2.1: **Room with {U} Frame**

Position is specified by giving the co-ordinates (x, y, z) of the origin of the obstacle frame in the {U} frame.

Orientation is specified by giving the three angles α, β, and γ where,
γ = angle of rotation about the X-axis of the {U} frame to get the obstacle frame
β = angle of rotation about the Y-axis of the {U} frame to get the obstacle frame
α = angle of rotation about the Z-axis of the {U} frame to get the obstacle frame

The above information gives us data to construct the transformation ma-

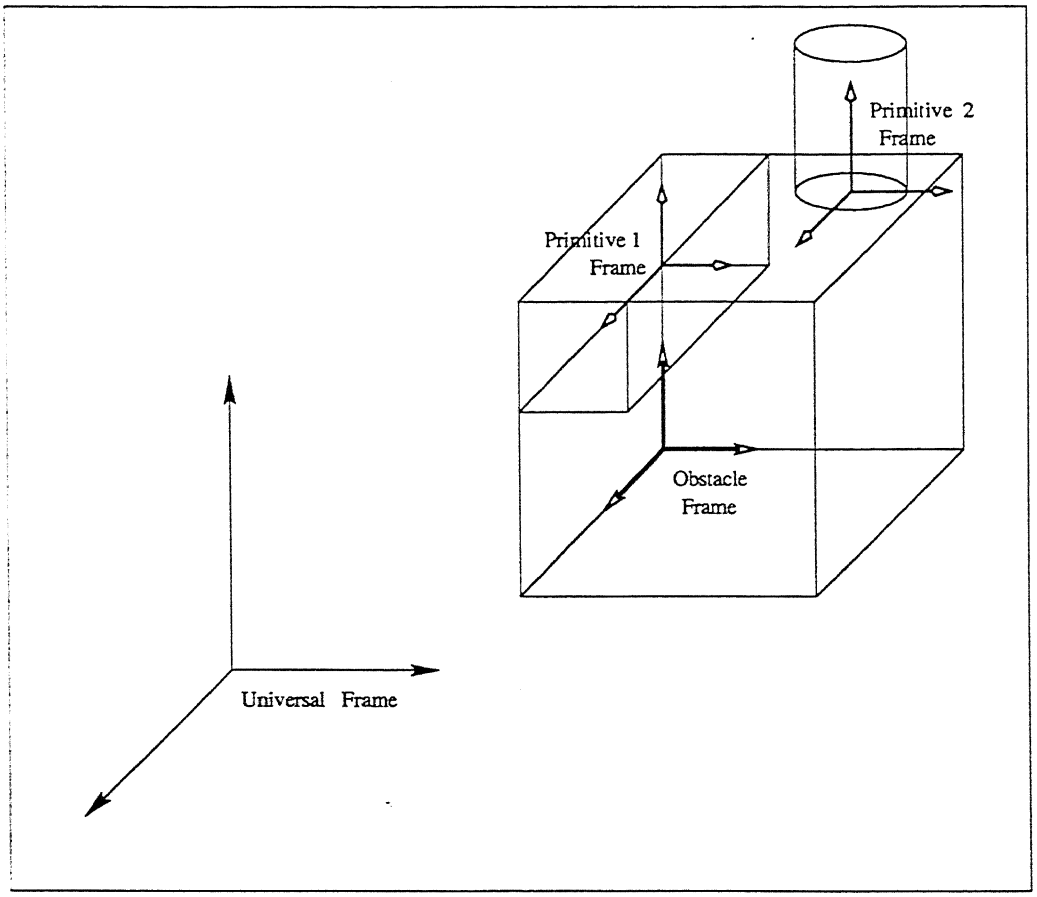


Figure 2.2: **Obstacle Frame Description**

trix ${}^{U}_{obs}T$ describing the obstacle frame with respect to the {U} frame.

$$ {}^{U}_{obs}T = \begin{bmatrix} c\alpha c\beta & c\alpha s\beta s\gamma - s\alpha c\gamma & c\alpha s\beta c\gamma + s\alpha s\gamma & x \\ s\alpha c\beta & s\alpha s\beta s\gamma + c\alpha c\gamma & s\alpha s\beta c\gamma - c\alpha s\gamma & y \\ -s\beta & c\beta s\gamma & c\beta c\gamma & z \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

Any point (vector) in the obstacle frame when pre-multiplied with this transformation matrix will give co-ordinates of that point in the {U} frame.
The obstacles are considered to be made of two kinds of primitives:

Parallelopiped which is in the form of a box.

Cylindrical which ranges from a solid cylinder to a sector of a hollow cylinder.

Any other shape of an obstacle needs to be constructed (in the current implementation) from primitives of these two kinds. A local frame is attached to each of the primitives called the primitive frame (Fig 2.2). Its complete description inside the obstacle is given by specifying the position and orientation of this primitive frame with respect to the obstacle frame to get the transformation matrix ${}^{obs}_{prim}T$. Thus any point inside the primitive can be expressed in the obstacle frame.

2.2 Manipulator Modeling

The manipulator is modeled as a set of links connected in a chain. It can be thought of as a set of bodies placed one after the other to form a long jointed arm. The links considered are of circular cross section and taper along the length.

Due to actuation considerations, manipulators are generally constructed from joints which exhibit just one degree of freedom. Most manipulators have **revolute joints** or have sliding joints called **prismatic joints**. Our manipulator consists of both these kinds of joints.

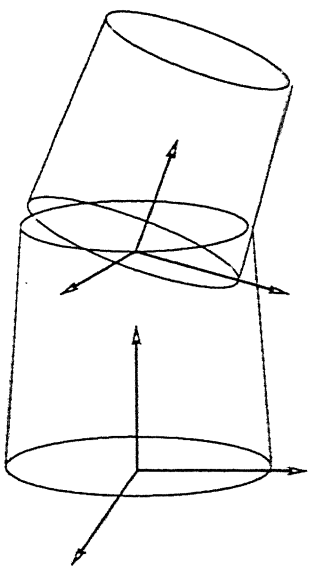


Figure 2.3: **Local Frames**

To describe the manipulator inside the room, a frame called $\{0\}$ frame is attached to the base of the manipulator. The position and orientation of this base frame is described by the transformation matrix ${}^{U}_{0}T$ constructed as described above.

To get the description of all links of the manipulator, local frames are attached to each of them[1] (Fig 2.3). The description of these link frames with respect to the preceding ones is given by the Denavit-Hartenberg (D-H) parameters. Using these D-H parameters we can construct link to link transformation matrices ${}^{i-1}_{i}T$.

The four D-H parameters are link twist (α_{i-1}), link length (a_{i-1}), link offset (d_i) and joint angle (θ_i). Of these, for a particular joint, three are fixed and one is variable. For a revolute joint, α_{i-1}, a_{i-1} and d_i are fixed while θ_i is the joint variable. For a prismatic joint, α_{i-1}, a_{i-1} and θ_i are fixed while d_i is the joint variable.

Link transformation is the transformation which defines frame $\{i\}$, attached to the i-th link, relative to the frame $\{i-1\}$, attached to the $(i-1)$-th link. For any given robot this transformation is a function of only one variable, the other three parameters being fixed by mechanical design. This transformation matrix is given by

$$
{}^{i-1}_{i}T = \begin{bmatrix} c\theta_i & -s\theta_i & 0 & a_{i-1} \\ s\theta_i c\alpha_{i-1} & c\theta_i c\alpha_{i-1} & -s\alpha_{i-1} & -s\alpha_{i-1}d_i \\ s\theta_i s\alpha_{i-1} & c\theta_i s\alpha_{i-1} & c\alpha_{i-1} & c\alpha_{i-1}d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}
$$

We can represent any link frame $\{n\}$ with respect to the base frame $\{0\}$ by applying the following transformation

$$
{}^{0}_{N}T = {}^{0}_{1}T\ {}^{1}_{2}T\ {}^{2}_{3}T\ \cdots\ {}^{N-1}_{n}T
$$

Since the description of $\{0\}$ frame is known with respect to $\{U\}$ frame, we can get the description of any frame n with respect to $\{U\}$ frame as

$$
{}^{U}_{N}T = {}^{U}_{0}T\ {}^{0}_{N}T
$$

[1]For frame assignment, refer Craig [12], a brief description is also included in Appendix B.

Thus the complete description of the manipulator is known inside the work-cell.

Chapter 3

Obstacle Mapping

There are various ways to plan paths for manipulators which are free of any kind of obstacles. One way is to do it in Cartesian space and the other in Configuration space.

3.1 Cartesian Space

Here we have a description of the workcell in the form of 3-D objects. Some of the methods which work here are the *Potential Field Method* and *Sensor based path planning*. The potential field method suffers from local minima and it is difficult to formulate a potential function which is free of local minima. Sensor based path planning employs considerable electronics component. For hyper-redundant manipulators the earlier method is prohibitively complex and the latter one is resource-dependent.

3.2 Configuration Space

The C-space approach due to Lozano-Pérez [1] is a relatively new method. Here, we map the whole workcell of the hyper-redundant robot into its C-space. After the mapping we have the description of the C-space obstacles. Thereafter the C-space can be searched for a feasible path. This mapping for hyper-redundant manipulators does not exist as yet and this is the first attempt to accomplish it.

Once the mapping is complete, we have a description of the C-space of the hyper-redundant manipulator in the form of discrete points[1]. These points belong to any one of the following types. (See Fig 3.1)

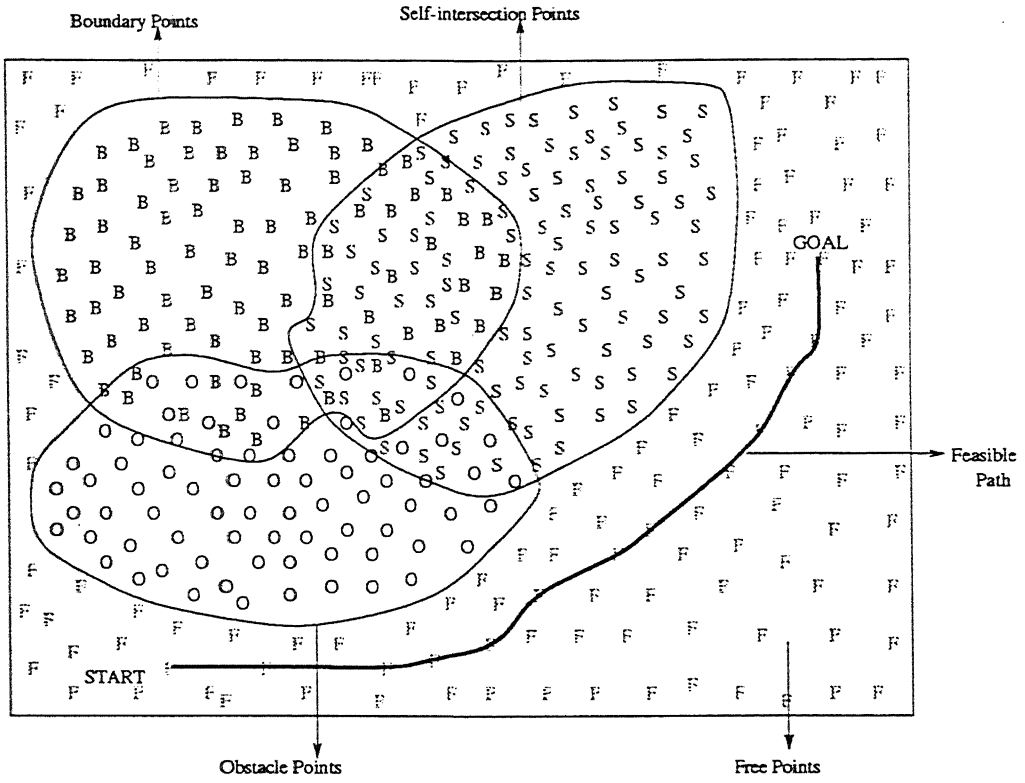


Figure 3.1: C-space Points for a Manipulator

Obstacle points These points are generated when any link of the manipulator collides with the physical obstacle present inside the workspace of the robot. They actually represent those configurations of the robot when it hits an obstacle.

Boundary points These are generated when the manipulator collides with the workcell boundary. They are those manipulator configurations when it hits the workcell boundary.

[1]Points in C-space are actually configurations that the manipulator takes in the 3-D Cartesian space.

Self-intersection points Apart from the above two types of C-space obstacles, a third type is this one. It represents those configurations of the manipulator when any link of it has collided with any other link of itself. Such C-space obstacles are not directly 'visible' in Cartesian space but they come into play when the robot takes different configurations.

Free points As the name indicates, these points represent those configurations of the hyper-redundant manipulator when it does not hit any obstacle or any link of itself. A feasible path has to pass through such free points.

3.3 Mapping

Mapping of obstacles from Cartesian space to Configuration space and identifying separate C-obstacles forms the major part of this work. This mapping is basically performed by generating a large number of configurations of the hyper-redundant manipulator and computing the intersections with the obstacles.

3.3.1 METHODOLOGY

The methodology can be enumerated in the following steps.

1. Input workcell in cell decomposed form.
2. Input manipulator using D-H parameters.
3. Generate a large number of configurations of the manipulator.
4. Compute intersections of the manipulator body with all kinds of obstacles.
5. Store configurations appropriately.
6. Identify separate C-obstacles.

3.3.2 Workcell input

For this work, the workcell that has been considered is in the form of a rectangular room[2]. The workcell consists of a number of obstacles located at various locations inside it (Fig 3.2).

ROOM

The dimensions of the room (length, height, breadth) are input by the user. Thereafter the room is divided into discrete equal-sized *cubic* cells (Fig 3.3). The number of cells depends on a *resolution factor*. A value of this factor greater than 1 increases the resolution of this cubic discretization, e.g. if factor = 2, the number of cells in each direction becomes double the value of the dimension in that direction.

Once the dimensions are input, the cells on the boundary of this room are flagged as **boundary**. The Universal frame ({U} frame) is attached to the far-bottom-left corner of this room. X-axis is from left to right, Y-axis is from bottom to top and Z-axis points right at the viewer.

OBSTACLES

The obstacles are basically made of two kinds of primitives: parallelopiped and cylindrical. For the parallelopiped primitive, the length, height and breadth are input along with its description with respect to the obstacle frame. For the cylindrical primitive, its inner and outer radii, start and end angles are input along with its length. Its description with respect to the obstacle frame is also input.

For each obstacle input, the description of the primitives is provided with respect to the local frame attached to the obstacle. The primitives can be located at various places inside the obstacle and at different orientations. The description of the obstacle is input in the form of position and orientation of its local frame with respect to the {U} frame.

[2]Arbitrary shape of workcell can be implemented by defining obstacles which touch the boundary and flagging cells belonging to these obstacles as boundary cells.

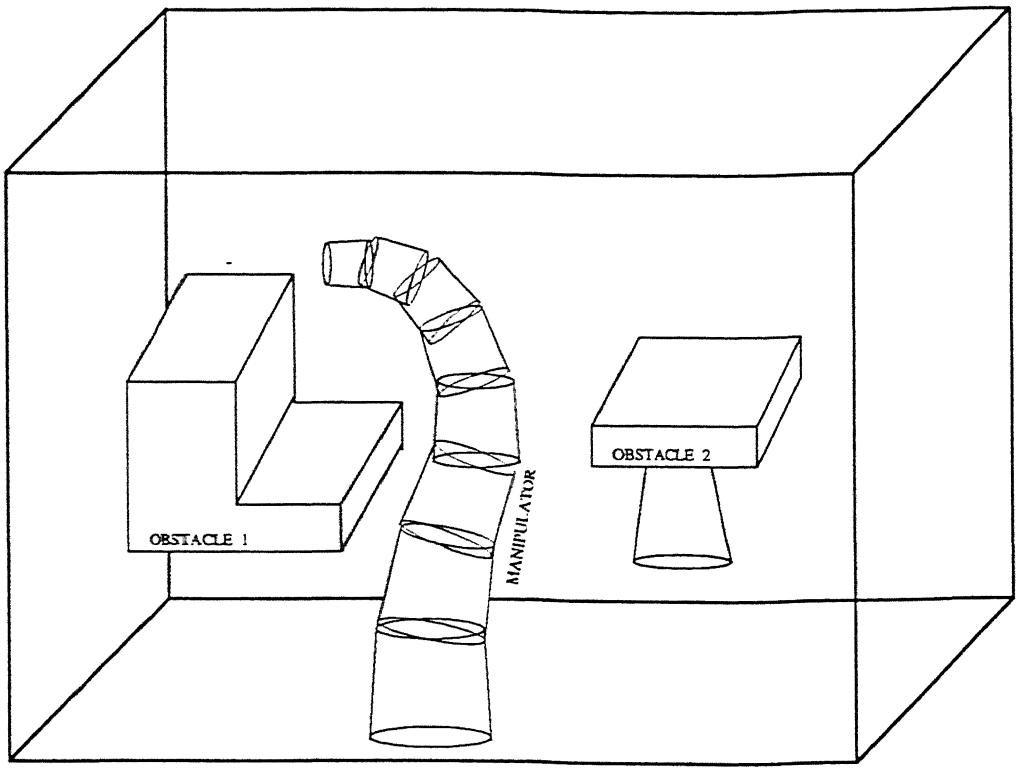


Figure 3.2: **A Typical Workcell**

Once the complete description of one obstacle is known, the cells belonging to the obstacle are flagged as **obs[1]**. Similarly, other obstacles are fed and corresponding cells are tagged as **obs[2]**, **obs[3]** etc. The remaining cells i.e. cells not belonging to either the boundary or any obstacle are flagged as **empty**.

3.3.3 Manipulator input

The manipulator is input in the form of D-H parameters. The four D-H parameters are link twist (α_{i-1}), link length (a_{i-1}), link offset (d_i) and joint angle (θ_i). Of these, for a particular joint, three are fixed and one is variable. For a revolute joint, α_{i-1}, a_{i-1} and d_i are fixed while θ_i is the joint variable. For a prismatic joint, α_{i-1}, a_{i-1} and θ_i are fixed while d_i is the joint variable.

For each joint, the lower and upper limits of the joint variable are also

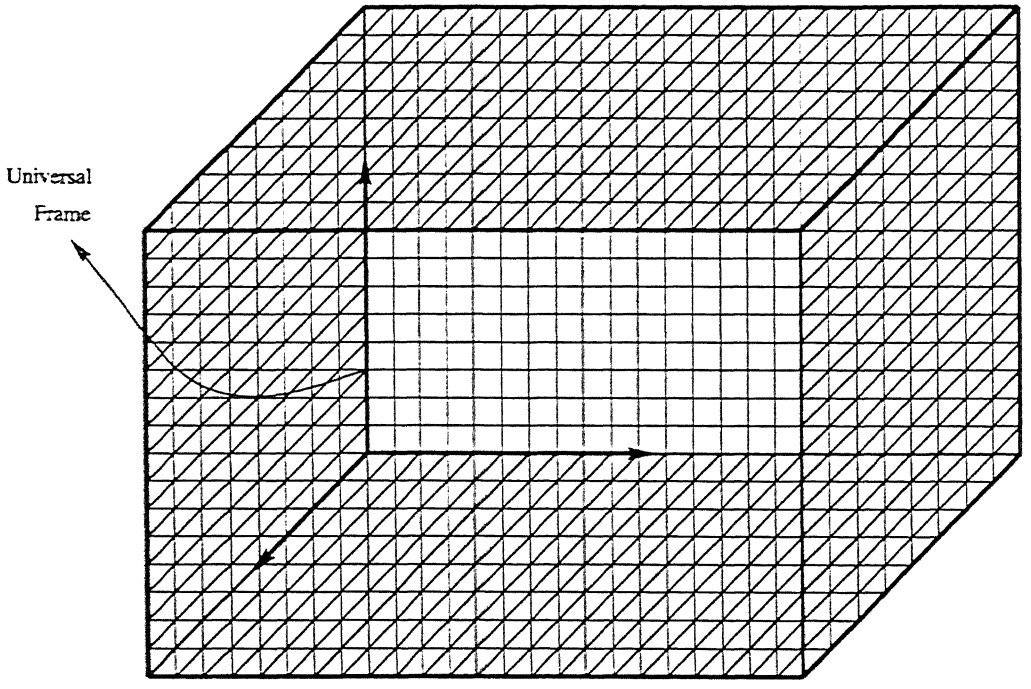


Figure 3.3: **Cell Decomposition of Workcell Room**

input. The whole manipulator is constructed by using this set of link parameters and a 'generated' joint variable. If a link is not along any of the X, Y or Z axes of its local frame, the user has to specify a set of points representing the 'mid-line' curve of the link.

The links considered in this work are of circular cross section and taper along their length. Physical lengths of the links are also input[3]. The user also specifies whether frame is located on the top or bottom of the link[4]. In addition to this information, the user also gives the base radius of the first link and the radius reduction ratio [5]. Flagging of cells, as belonging to manipulator, is done later when configurations are generated.

[3]This is required in the case when two succesive links share the same joint axis and local frame of one is located at its base and of the other at its top. In such case, if the physical length of the first link is not given, we lose the information where that link ends and the other begins.

[4]This is to ascertain the direction in which the link is to be grown.

[5]This value determines to what extent the radius will reduce along the length of the link, i.e. it gives the taper.

The description of the **fixed** *base frame* attached to the base of the first link is also input from the user. We call it the {0} frame.

3.3.4 Generating configurations

After the workcell and the manipulator have been input, a large number of configurations of the hyper-redundant manipulator are generated. This is done by assigning different values to joint variables of all the joints. The user inputs a value for the number of 'steps'. Depending on this value, a fixed number of divisions are made between the specified lower and upper limits of the joint variables. Each joint variable is made to take all the values at these divisions, for each new value of its previous joint variable. In this way, the workspace of the robot is explored exhaustively.

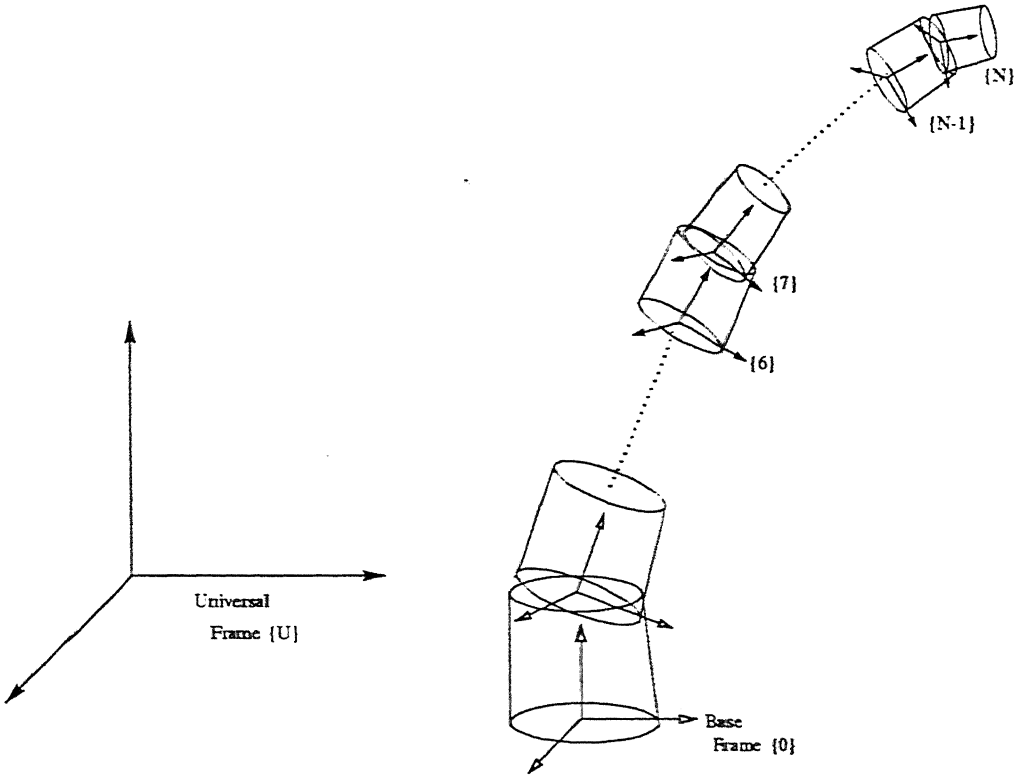


Figure 3.4: **Link to Link Frame Transformations**

3.3.5 Computing intersections

For each configuration generated by the above process, it is checked whether the manipulator body intersects with any of the obstacles. This is done by computing intersections of the manipulator with the workcell boundary, workcell obstacles and with its own links.

Refer Figure 3.4. Here the {U} frame has been shown attached to the far-bottom-left corner of the room. The {0} frame is the one attached to the base of the first link. A few intermediate links and the end links have also been shown with their local frames attached.

Link transformations

For any given robot the link to link transformation matrix is given by

$$ {}^{i-1}_{i}T = \begin{bmatrix} c\theta_i & -s\theta_i & 0 & a_{i-1} \\ s\theta_i c\alpha_{i-1} & c\theta_i c\alpha_{i-1} & -s\alpha_{i-1} & -s\alpha_{i-1} d_i \\ s\theta_i s\alpha_{i-1} & c\theta_i s\alpha_{i-1} & c\alpha_{i-1} & c\alpha_{i-1} d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

Once the description of any link frame with respect to its previous link frame is available, we can represent any link frame $\{n\}$ with respect to the base frame {0} by applying the following transformation

$$ {}^{0}_{N}T = {}^{0}_{1}T \, {}^{1}_{2}T \, {}^{2}_{3}T \cdots {}^{N-1}_{n}T $$

Since the description of {0} frame is known with respect to {U} frame, we can get the description of any frame $\{n\}$ with respect to {U} frame as

$$ {}^{U}_{N}T = {}^{U}_{0}T \, {}^{0}_{N}T $$

Now, we know the co-ordinates of any point on the boundary of the link-body in the link frame since the physical dimensions of the link, like base radius and its length, are known (from user input). Applying the transformation ${}^{U}_{N}T$ we get the same co-ordinates with respect to the {U} frame. Thus we have the address of the cells belonging to the boundary of a particular link of the manipulator with respect to the {U} frame. These cells are flagged as **lnk**[i] belonging to the i-th link of the manipulator. In this way

cells belonging to the boundary of all the links of the manipulator are flagged. Now, since the cells belonging to the boundary of any link of the manipulator are known, it is checked whether they belong to a workcell obstacle, workcell boundary or any other link-boundary of the manipulator.

3.3.6 Storing configurations

As and when the intersections are detected, they are stored in separate data files depending on the type of obstacle. Configurations for which no intersections are detected are stored as 'free' configurations. A feasible path passes through these free configurations only.

It may be observed that, through the above method, we get all points lying inside the C-obstacle while only boundary points would suffice for representation of the C-obstacles. In this context, it may be mentioned that getting boundary points of the C-obstacles by identifying configurations, where link-boundaries are tangent to any of the obstacles, requires a great deal of effort on the part of the programmer[6], whereas it is quite simple to identify a boundary point by making sure that all its $3^n - 1$ neighbors are not C-obstacle points. This identification of boundary points can be made by carrying out an independent search among all the C-obstacle points.

3.3.7 Identifying separate C-obstacles

As and when an obstacle point is encountered, a check is made in the n-dimensional C-space whether any of its neighbours has been encountered (as an obstacle point) before or not. If yes, the concerened point is included in the same C-obstacle, else it is taken to be the first point of the next C-obstacle. Though this method makes sure that the obtained C-obstacles do not overlap, occasionally it is possible that a single C-obstacle may be reported as two or more.

For an n-dimensional point $P(x_1, x_2, \ldots, x_n)$ all points between $P(x_1 - 1, x_2 - 1, \ldots, x_n - 1)$ and $P(x_1 + 1, x_2 + 1, \ldots, x_n + 1)$ are taken to be its neighbours i.e. each n-dimensional point has $3^n - 1$ neighbours.

[6]or the user of the output of the present work in any context

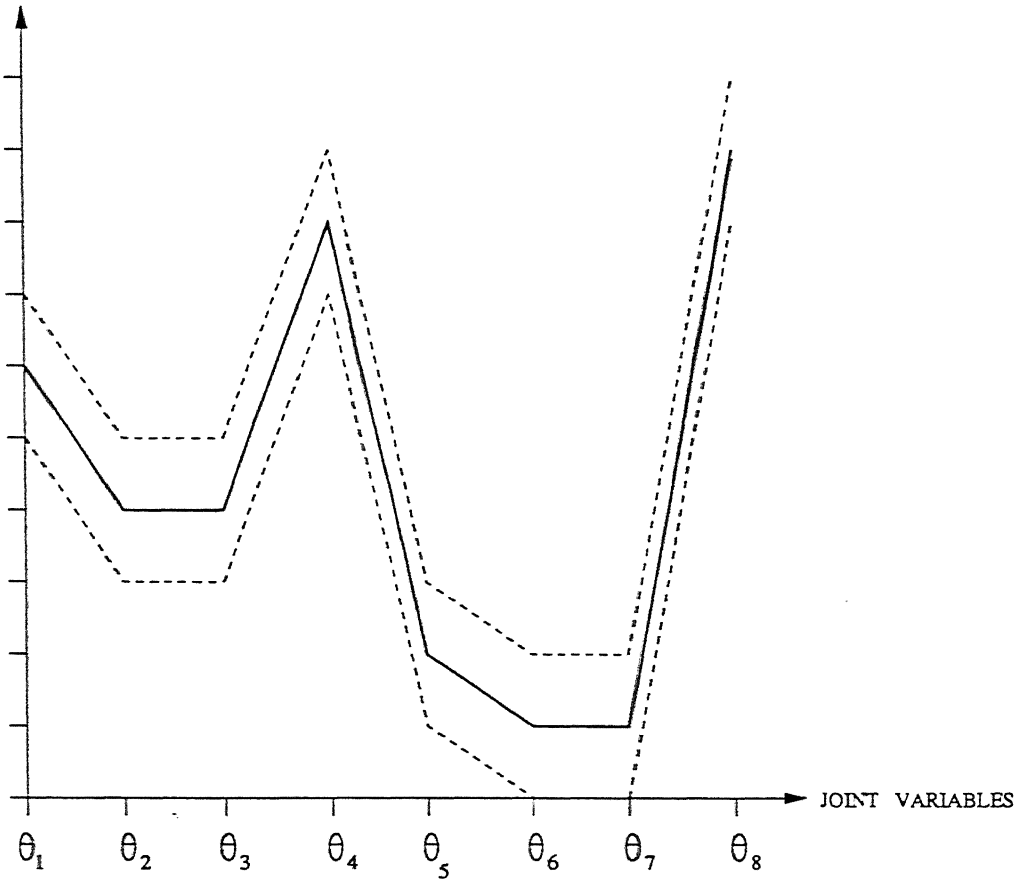


Figure 3.5: **Neighbours in C-space**

In Figure 3.5, the thick line represents a configuration. All configurations lying between the dashed lines are taken to be neighbours for this configuration.

All separate C-obstacles are reported in different (output) data files.

3.3.8 Algorithm

Purpose: Mapping obstacles from Cartesian space to Configuration space

Input: Room

- dimensions

- resolution factor

Obstacle

- description
- position
- orientation

Manipulator

- D-H parameters
- physical dimensions of links
- position and orientation of base

Output: (i) Points in C-space representing obstacles and free configurations.

(ii) C-obstacles

Steps:

1. Divide room into equal sized cubes.

2. Get point on room boundary $\begin{bmatrix} x & y & z \end{bmatrix}^T$
   flag cell(x, y, z) = boundary

3. for (each obstacle)
   (i) for (whole body of each primitive)
   (a) get point on its boundary in its local frame

$$Pt_{\{prim\}} = \begin{bmatrix} x & y & z \end{bmatrix}^T$$

   (b) transform point to {U}

$$Pt_{\{U\}} = {}^{U}_{obs}T \, {}^{obs}_{prim}T \, Pt_{\{prim\}}$$

   (c) flag cell(x, y, z) = obstacle

4. for (each cell)
   (i) if (flag cell$(x, y, z) \neq$ boundary and flag cell$(x, y, z) \neq$ obstacle)

flag cell(x, y, z) = empty

5. $i \leftarrow 1$
call function recursive(i); (defined below)

(i) for (joint i)
(a) for ($b = 0$ to $b =$ steps)

$$\theta_i = \theta_{lower} + b\left(\frac{\theta_{upper} - \theta_{lower}}{steps}\right)$$

(b) if ($i <$ total number of joints)
call function recursive($i + 1$);
(c) for ($i = 1$ to $i =$ total number of joints)
calculate matrix ${}^{U}_{i}T$
(d) for (whole body of link i)
(1) get a point on its boundary in its local frame
(2) transform point to {U}

$$Pt_{\{U\}} = {}^{U}_{i}T\ Pt_{\{i\}} = \begin{bmatrix} x & y & z \end{bmatrix}^T \text{ (say)}$$

(3) if (flag cell(x, y, z) = boundary)
report intersection
(4) if (flag cell(x, y, z) = obstacle)
report intersection
(5) if ($i \geq 2$)
if (flag cell(x, y, z) = any of link[0] to link[i-2])
report intersection
(6) if (flag cell(x, y, z) = empty)
flag cell(x, y, z) = link[i]
(7) if (intersection reported)
(a) if (neighbour in C-space is obstacle)
include in same C-obstacle
(b) else
include in next C-obstacle

(8) if (intersection not reported)
report no intersection
(e) for ($i = 1$ to $i =$ total number of joints)
if (cell(x, y, z) = link$[i]$)
flag cell(x, y, z) = empty

Chapter 4

Inverse Kinematics

The problem of solving the kinematic equations of a manipulator is a non-linear one. Given the numerical value of ${}^0_N T$ we attempt to find values of $\theta_1, \theta_2, \ldots, \theta_n$.

For the case of an arm with 6 degrees of freedom we have 12 equations and 6 unknowns. However from the 9 equations arising from the rotation matrix portion of ${}^0_6 T$, only 3 equations are independent. These added with 3 equations from the position vector portion of ${}^0_6 T$ give 6 equations with 6 unknowns. These equations are non-linear, transcendental equations which can be quite difficult to solve.

For the case of manipulators with still higher degrees of freedom, closed form solutions are not possible.

4.1 Hyper-Redundant Manipulators

The problem of inverse kinematics is stated as follows:

> Given a particular position and orientation of the end-effector of the manipulator, find the sets of joint variables which give rise to that particular position and orientation of the end-effector.

For hyper-redundant manipulators, the degrees of freedom is quite high, so infinite solutions exist. In this work, we use GA to solve the inverse kinematics problem applying the concept of 'niching and speciation'.

4.2 Genetic Algorithms

Genetic algorithms are search algorithms based on the mechanics of natural selection and natural genetics. They combine survival of the fittest among string structures with a structured yet randomized information exchange to form a search algorithm with some of the innovative flair of human search. In every generation a new set of artificial creatures (strings) is created using bits and pieces of the fittest of the old; an occasional new part is tried for good measure. While randomized, GAs efficiently exploit historical information to speculate on new search points with expected improved performance.

The GA is based on two biological phenomena: natural selection and sexual recombination. In this, the possible solutions to a problem are represented as bit strings. The GA maintains a population of bit strings, and reproduces them preferentially based on quality of solutions they represent. This is analogous to natural selection.

A simple genetic algorithm that yields good results in many practical problems is composed of three operators:

1. Reproduction
2. Crossover
3. Mutation

Reproduction is a process in which individual strings are copied according to their objective function values, f (also called fitness function). Intuitively we can think of the function f as some measure of profit or goodness that we want to maximize. Copying strings according to their fitness values means that strings with a higher value have a higher probability of contributing one or more offspring in the next generation[1]. Once a string has been selected for reproduction, an exact replica of the string is made. This string is then entered into a mating pool, a tentative new population, for further genetic operator action.

[1]This operator, of course, is an artificial version of natural selection.

Crossover may proceed in two steps after reproduction. First, members of the newly reproduced strings in the mating pool are mated at random. Next, each pair of strings undergoes crossing over as follows: an integer position k along the string is selected uniformly at random between 1 and the string length less one $[1, l-1]$. Two new strings are created by swapping all characters between positons $k+1$ and l inclusively.

Mutation plays a small but important role in genetic methods: it prevents the population from becoming completely uniform. As the optimization process progresses, the population improves by converging on a relatively small area of the search space. this necessarily means that the population becomes less diverse. Mutation introduces small changes which allow optimization to explore itself. When the population is highly fit, most mutations are likely to be detrimental. However, they can expand the search space beyond what would be explored by crossover alone and this can occasionaly be beneficial.

Genetic approach does not rely on problem-specific features. They have two basic requirements: possible solution should be represented in a way such that two parents can be combined to yield an offspring, and a method of evaluating solutions must be supplied. These properties make GA attractive for our work.

4.3 Problem Formulation

In this work, we are required to perform inverse kinematics using GA. We have framed this as an unconstrained optimization problem. The variables in this problem are the joint variables that we are trying to solve for. The values of these variables in the final population will give the required joint variables for the specified end-effector position and orientation.

4.3.1 Objective function

Let ${}^{U}_{end}T_{req}$ be the required description of the end-effector frame with respect to the {U} frame and ${}^{U}_{end}T_{GA}$ be the description based on the values of joint

variables for an individual at a particular generation during the execution of the GA. The objective function has been framed as the difference between the values of the elements of these two matrices. A scaling factor has been provided for the elements belonging to the rotation matrix part of these transformation matrices.

If a_{ij} be an element belonging to the matrix ${}^{U}_{end}T_{req}$ and b_{ij} be an element belonging to the matrix ${}^{U}_{end}T_{GA}$, the objective function is:

$$f = k\sum_{i=1}^{3}\sum_{j=1}^{3}|a_{ij} - b_{ij}| + \sum_{i=1}^{3}|a_{i4} - b_{i4}|$$

where k is the scale factor determined experimentally. Thus, this is a minimization problem.

The fitness function that we want to maximize is given below:

$$fitness = \frac{1}{1+f}$$

4.4 Niching and Speciation

In our problem, because of the hyper-redundancy, the function space is highly multimodal, i.e. we have infinite number of peaks. In such a scenario, it is very likely that the GA will converge to a single peak i.e. majority of the individuals in the final population will belong to any one of the numerous peaks. In a general case, we are interested in finding as many solutions as possible and that too in a diverse space. For this we apply the concept of niching and speciation.

The specialization permitted by sexual differentiation is carried further in nature through speciation and niche exploitation. Intutively we may view a niche as an organism's job or role in an environment, and we can think of a species as a class of organism with common characterstics.

In case of multimodal problems (see Fig 4.1), if we start from an initial population chosen uniformly at random, we obtain a relatively even spread of points across the function domain. As reproduction, crossover and mutation proceed, the population climbs the hills, ultimately distributing most of the strings near the top of one hill among the several. This ultimate convergence

on one peak or another without differential advantage is caused by genetic drift - stochastic errors in sampling caused by small population sizes. (Refer page 185 of Goldberg [14])

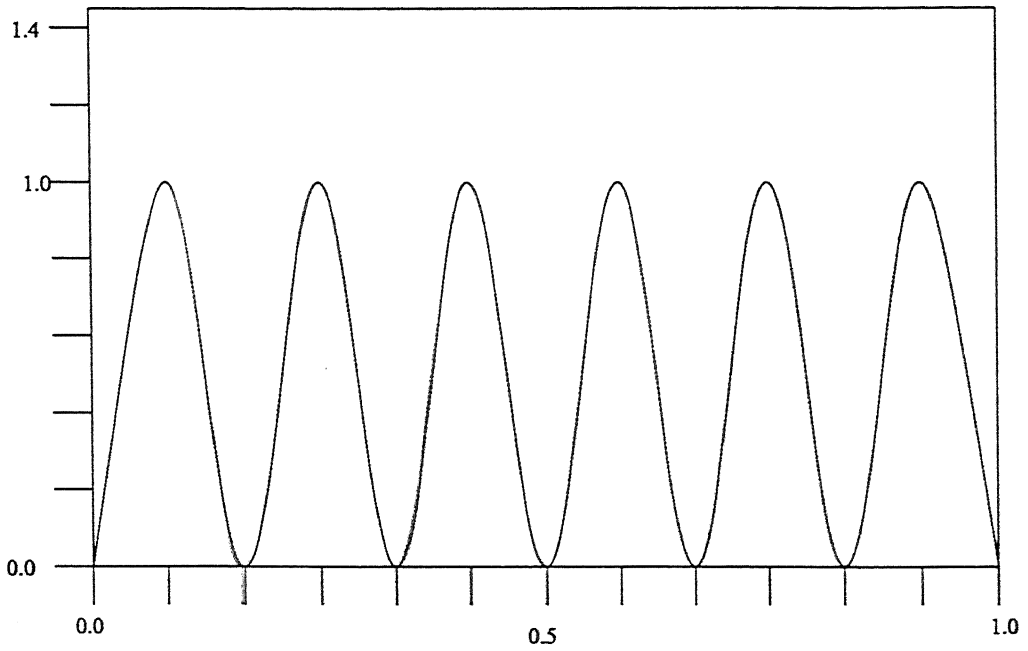


Figure 4.1: **Equal Peaks**

4.4.1 Theory of niche and species

In such multimodal problems, instead of assigning the usual fitness to all individuals, each individual is forced to **share** the fitness of all the other individuals in its species. The incorporation of forced sharing discourages crowding of the population at a particular peak and causes the formation of stable sub-populations (species) at different peaks (niches). Furthermore, the number of individuals devoted to each niche is proportional to the expected niche payoff.

A practical scheme that directly uses the sharing metaphor to induce niche and species is detailed by Goldberg and Richardson [15]. In this scheme, a *sharing function* is defined to determine the neighbourhood and degree of sharing for each string in the population.

4.4.2 Sharing function method

For a given individual the degree of sharing is determined by summing the sharing function values contributed by all other strings in the population. Strings close to an individual require a high degree of sharing (close to one), and strings far from the individual require a very small degree of sharing (close to zero). After accumulating the total number of shares in this manner, an individual's derated fitness is calculated by taking the potential fitness (the unshared value) and dividing through by the accumulated number of shares.

Given a set of n_k solutions in the k-th generation [16] each having a dummy fitness value f_k, the sharing procedure is performed in the folowing way for each solution $i = 1, 2, \ldots, n_k$:

Step 1: Compute a normalized Euclidean distance measure with another solution j in the k-th generation as follows:

$$d_{ij} = \sqrt{\sum_{p=1}^{P} \left(\frac{x_p^{(i)} - x_p^{(j)}}{x_p^u - x_p^l} \right)^2},$$

where P is the number of variables in the problem. The parameters x_p^u and x_p^l are the upper and lower bounds of variable x_p.

Step 2: This distance d_{ij} is compared with a pre-specified parameter σ_{share} and the following *sharing function* value is computed:

$$Sh(d_{ij}) = \begin{cases} 1 - \left(\frac{d_{ij}}{\sigma_{share}}\right)^2, & \text{if } d_{ij} \leq \sigma_{share}, \\ 0, & \text{otherwise.} \end{cases}$$

Step 3: Increment j. If $j \leq n_k$ go to Step 1 and calculate $Sh(d_{ij})$. If $j > n_k$, calculate niche count for i-th solution as follows:

$$m_i = \sum_{j=1}^{n_k} Sh(d_{ij})$$

Step 4: Degrade the dummy fitness f_k of i-th solution in the k-th generation to calculate the shared fitness f_i', as follows:

$$f_i' = \frac{f_k}{m_i}$$

This procedure is continued for each $i = 1, 2, \ldots, n_k$ and a corresponding value f_i' is found.

The above sharing procedure requires a pre-specified parameter σ_{share} [16] which can be calculated as follows:

$$\sigma_{share} \approx \frac{0.5}{\sqrt[p]{10}}$$

Thus when many individuals are in the same neighbourhood, they contribute to one another's share count, thereby derating one another's fitness values. As a result, this mechanism limits the uncontrolled growth of particular species within a population.

4.5 Solution Clusters

It has been said before that because of hyper-redundancy, the function space is highly multimodal and the sharing function method is used to get stable sub-populations at different peaks. What it means is that most of the individuals in the population which we will have in the final generation will be in 'clusters' or crowds. It is expected that many solutions will have θ's which differ by a small amount, i.e. there will be a cluster around a particular value of θ. There will be a number of such clusters. Some stray individuals are also expected because of mutation.

Chapter 5

Results

Computer programs were written for:

1. (a) Mapping obstacles
   (b) Getting separate C-obstacles
2. Getting inverse kinematic solutions

The programs have been written in C++ for *mapping obstacles* and *getting separate C-obstacles.* For inverse kinematics solution, source code for a real-coded simple genetic algorithm was obtained from the Kanpur Genetic Algorithms Lab (KanGAL), Deptt. of Mech. Engg., IIT Kanpur. The objective function was coded, and *niching and speciation* was incorporated using the Sharing Function.

The programs have been run for basically two types of hyper-redundant manipulators: **planar and spatial**. Although the results are shown for one of each type, the program is a general one and can be used for mapping obstacles for a manipulator having any number of degrees of freedom.

5.1 Description of Figures

Although it is not possible to show the complete description of C-space since it is n-dimensional and it is not possible to show an n-dimensional point on a 2-D graph, we have tried to give some idea of the C-space by plotting two

two kinds of plots. To illustrate the idea, an example of a planar 3-DOF manipulator is presented here. The workcell is roughly shown in Fig 5.1.

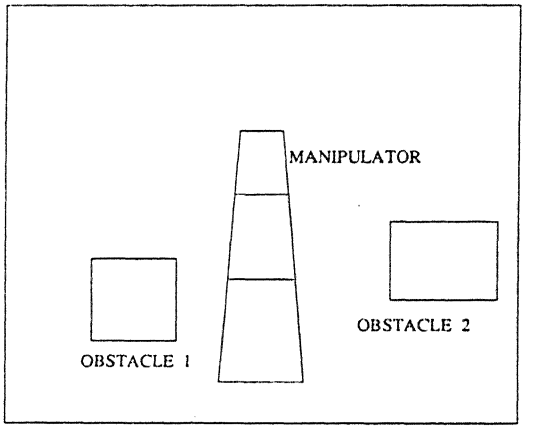


Figure 5.1: **Workcell for 3-DOF manipulator**

Although the manipulator is not redundant, the plots have been presented to convey some idea about the C-space. One (Polyline type) is for depicting boundary points, self-intersection points, obstacle points and free points. The other (Projection type) is used for showing separate C-obstacles.

5.1.1 Polyline plot

Here we plot each n-dimensional point as a polyline. The joint variables themselves have been shown along the X-axis (located at equal intervals) and their magnitude along the Y-axis. The magnitudes have been normalized as:

$$\theta_i = \frac{\theta_i - \theta_{lower}}{\theta_{upper} - \theta_{lower}}$$

For presentation purposes, the interval between the limits of the joint variables has been kept low.

Each polyline starting from θ_1 through θ_n represents one configuration of the manipulator (or one point in the n-dimensional C-space). For each type of obstacle we get a series of such configurations which have been shown as plots. Plots of free configurations have also been shown. The plots have

been drawn in OpenGL. Fig 5.2 shows the polyline plot for the planar 3-DOF manipulator. Here three C-obstacles have been shown.

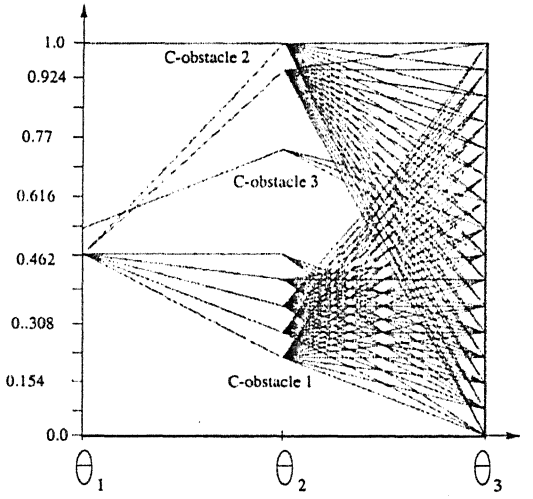


Figure 5.2: **Polyline plot for 3-DOF manipulator**

One limitation of this output plot is that with large number of data points and/or complicated shape of the C-obstacle, the plots become extremely jumbled, defying any interpretation. In such a situation an indirect assessment can be made by the following alternative output.

5.1.2 Projection plot

Here we have tried to use a concept akin to isometric projection. The axes are located at equal angles to each other where angle $a = \frac{2\pi}{n}$ (n = degrees of freedom). The co-ordinates (x, y) of any n-dimensional point on this 2-D graph is given by

$$\left\{ \begin{array}{c} x \\ y \end{array} \right\} = \left\{ \begin{array}{l} \theta_1 \cos(0) + \theta_2 \cos(a) + \theta_3 \cos(2a) + \theta_4 \cos(3a) + \cdots + \theta_n \cos((n-1)a) \\ \theta_1 \sin(0) + \theta_2 \sin(a) + \theta_3 \sin(2a) + \theta_4 \sin(3a) + \cdots + \theta_n \sin((n-1)a) \end{array} \right\}$$

Thus, each n-dimensional point is represented here as a projected 2-D point. The θ values are normalized.

The idea here is to have some a posteriori verification that C-space points reported as a C-obstacle are actually contiguous. It can be argued that originally non-contiguous points also may appear contiguous in the projection.

To alleviate such possibilities, various projections are tried by cyclic permutations of the axes. Contiguity of points in all these projections gives a strong evidence of the contiguity of the original points. Fig 5.3 is the projection plot for the planar 3-DOF manipulator. Two plots have been shown as just two permutations of the axes are possible.

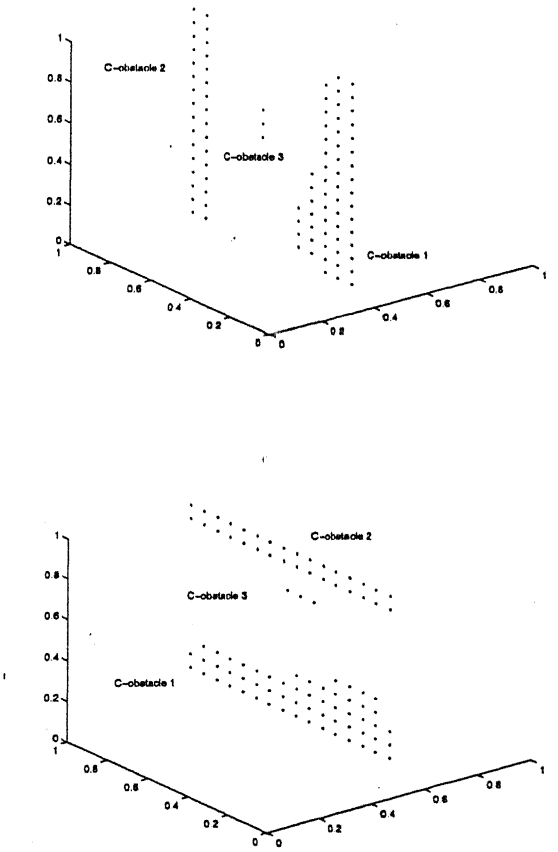


Figure 5.3: **Projection plots for 3-DOF manipulator**

For higher dimensional cases, we have shown such plots for 4 permutations of the n axes. The index for the axes have been shown in the figure. The plots have been drawn in MATLAB.

5.2 Mapping Obstacles and Getting C-obstacles

For mapping obstacles, the program has been run for a 5-DOF planar manipulator and an 8-DOF spatial hyper-redundant manipulator. For each type, we show the results for two different workcells. The separate C-obstacles have been shown for planar cases only.

For planar cases, the implementation is in spatial form itself where the third dimension (breadth), wherever it appears, is a dummy one.

5.2.1 Planar 5-DOF hyper-redundant manipulator

Description of manipulator

Joint	Type	α_{i-1}	a_{i-1}	d_i	θ_i	Length	Limits
1	Revolute	0	0	0	θ_1	8.0	10 350
2	Revolute	0	8.0	0	θ_2	8.0	10 350
3	Revolute	0	8.0	0	θ_3	8.0	10 350
4	Revolute	0	8.0	0	θ_4	8.0	10 350
5	Revolute	0	8.0	0	θ_5	8.0	10 350

Base radius of base link : 4.0
Radius reduction ratio : 0.997

Workcell 1

Dimensions (length, height, breadth) : 100 100 100
Resolution Factor : 1.0

Number of obstacles : 2

For obstacle 1
Frame origin co-ordinates wrt {U} frame : 5 5 5
Frame orientation wrt {U} frame (RPY angles) : 0 0 0
Number of primitives : 2
For primitive 1
Type : parallelopiped
Length : 10

Height : 5
Breadth : 8
Frame origin co-ordinates wrt obstacle frame : 3 0 3
Frame orientation wrt obstacle frame (RPY angles) : 0 0 0
For primitive 2
Type : cylindrical
Inner Radius : 2
Outer Radius : 5
Start Angle : 40
End Angle : 280
Length : 15
Frame origin co-ordinates wrt obstacle frame : 8 5 6
Frame orientation wrt obstacle frame (RPY angles) : 0 0 0

For obstacle 2
Frame origin co-ordinates wrt {U} frame : 35 5 5
Frame orientation wrt {U} frame (RPY angles) : 0 0 0
Number of primitives : 1
For primitive 1
Type : parallelopiped
Length : 5
Height : 5
Breadth : 5
Frame origin co-ordinates wrt obstacle frame : 0 0 0
Frame orientation wrt obstacle frame (RPY angles) : 0 0 0

Position and orientation of manipulator
Frame origin co-ordinates wrt {U} frame : 25 10 10
Frame orientation wrt {U} frame (RPY angles) : 0 0 0

Number of steps between limits of joint variables : 6

The given workcell is roughly drawn in Fig 5.1

Total C-obstacles detected = 15.
Here, some of the significant C-obstacles have been shown.

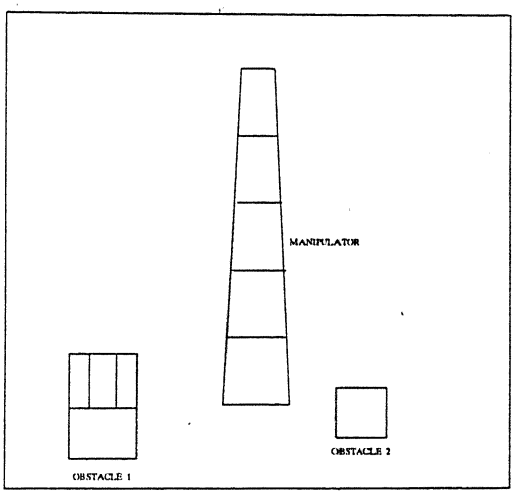


Figure 5.4: **Workcell 1 (5-DOF, Planar)**

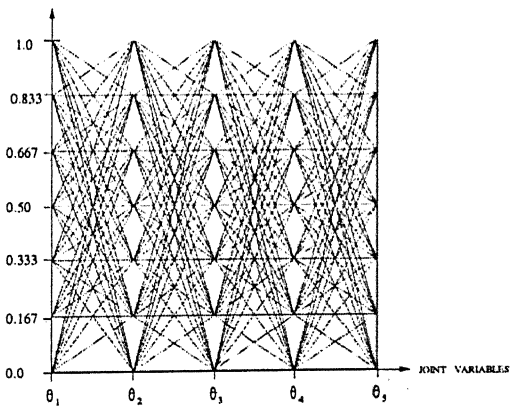


Figure 5.5: **5-DOF Workcell 1 Boundary Configurations**

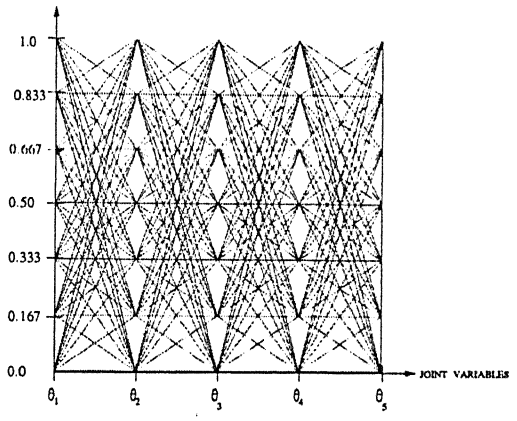


Figure 5.6: **5-DOF Workcell 1 Self-intersection Configurations**

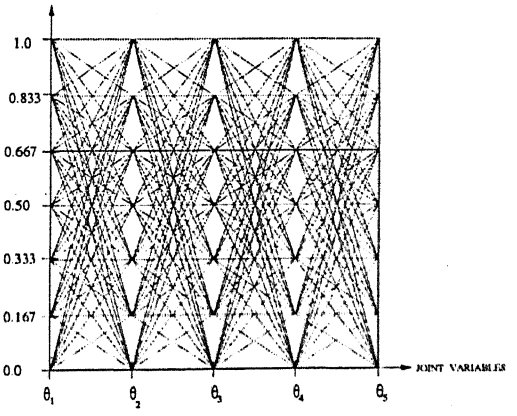


Figure 5.7: **5-DOF Workcell 1 Obstacle 1 Configurations**

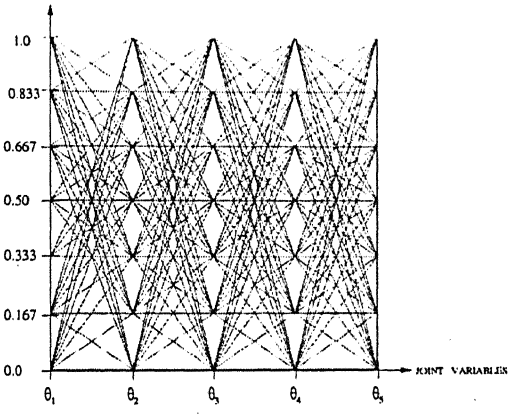


Figure 5.8: **5-DOF Workcell 1 Obstacle 2 Configurations**

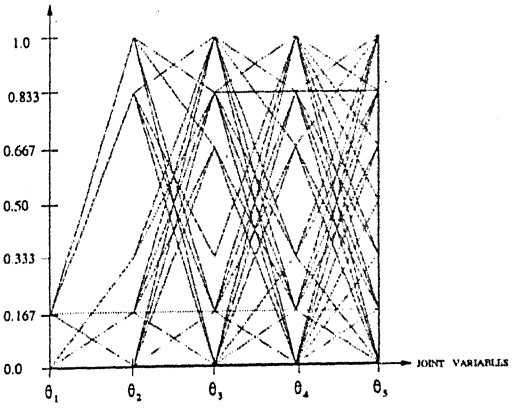


Figure 5.9: **5-DOF Workcell 1 Free Configurations**

The polyline plots show the different obtained configurations. From the plots it is evident that there are a lot many configurations of the manipulator in which it hits all kinds of obstacles, while the free configurations are much less. The plot for free configurations is sparse as compared to the other ones.

It must also be noted that although plots for different cases may look the same, they are not because a lot many combinations of θ's overlap each other and cannot be told from each other. Thus, as told earlier, it is not a complete representation and is just an attempt to convey some idea of the C-space, especially the connectedness of individual C-obstacles.

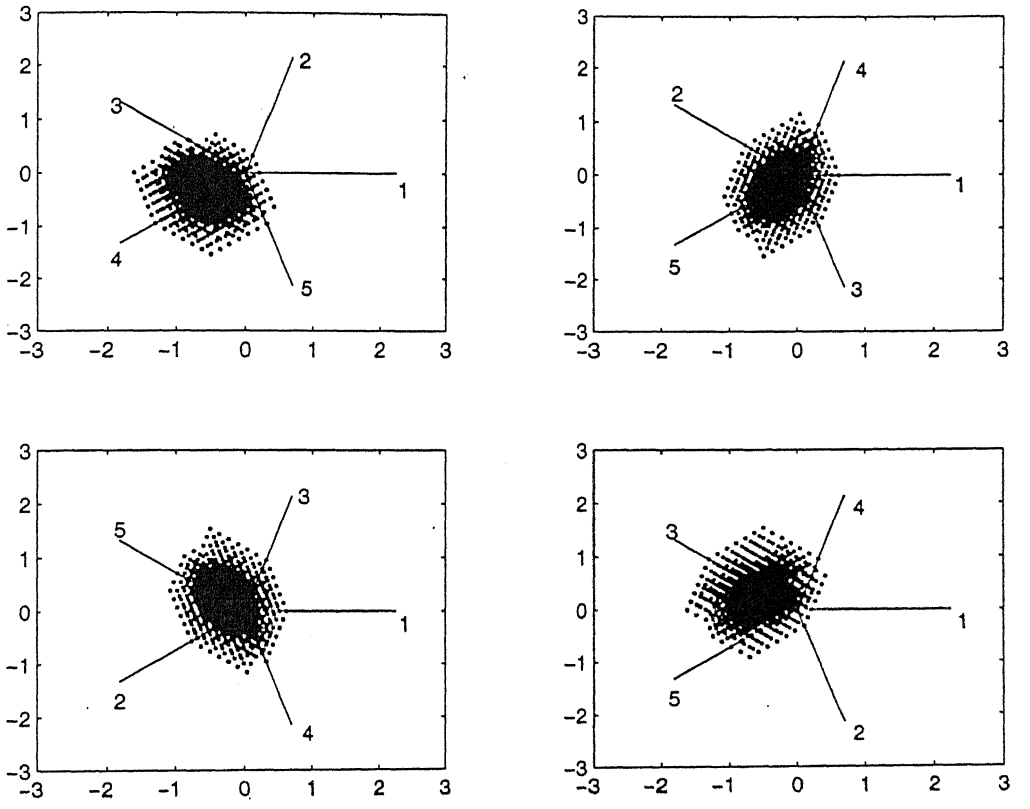


Figure 5.10: **5-DOF Workcell 1 C-obstacle 1**

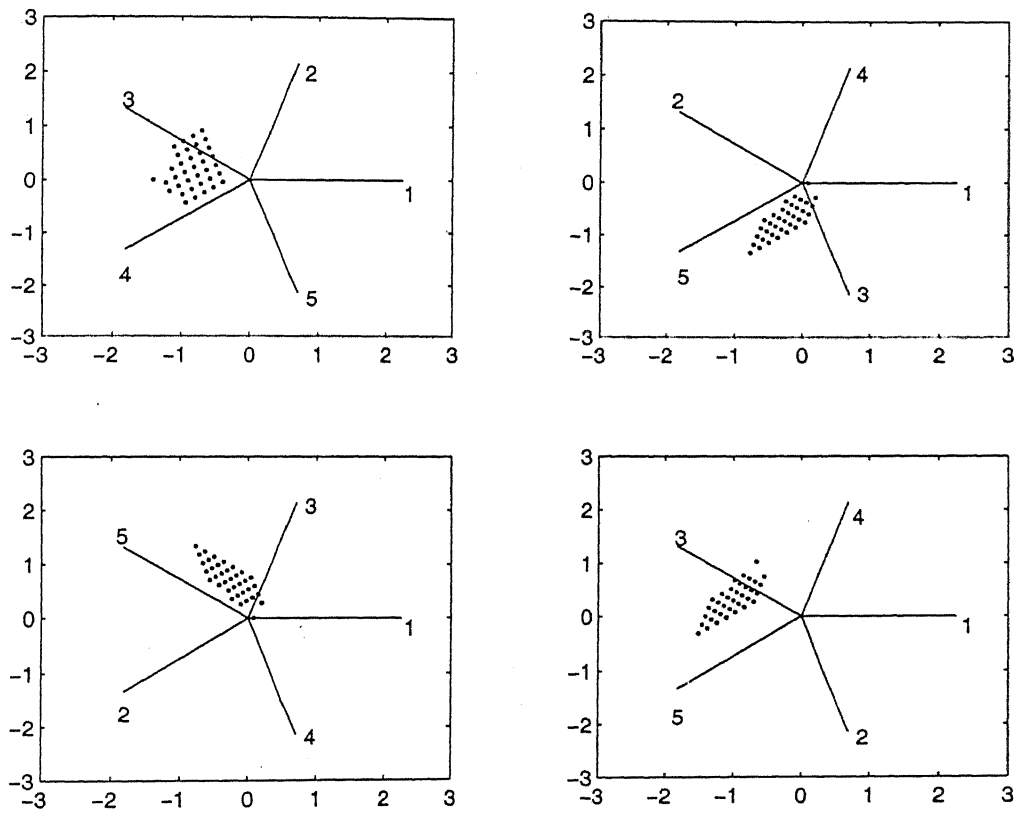


Figure 5.11: **5-DOF Workcell 1 C-obstacle 2**

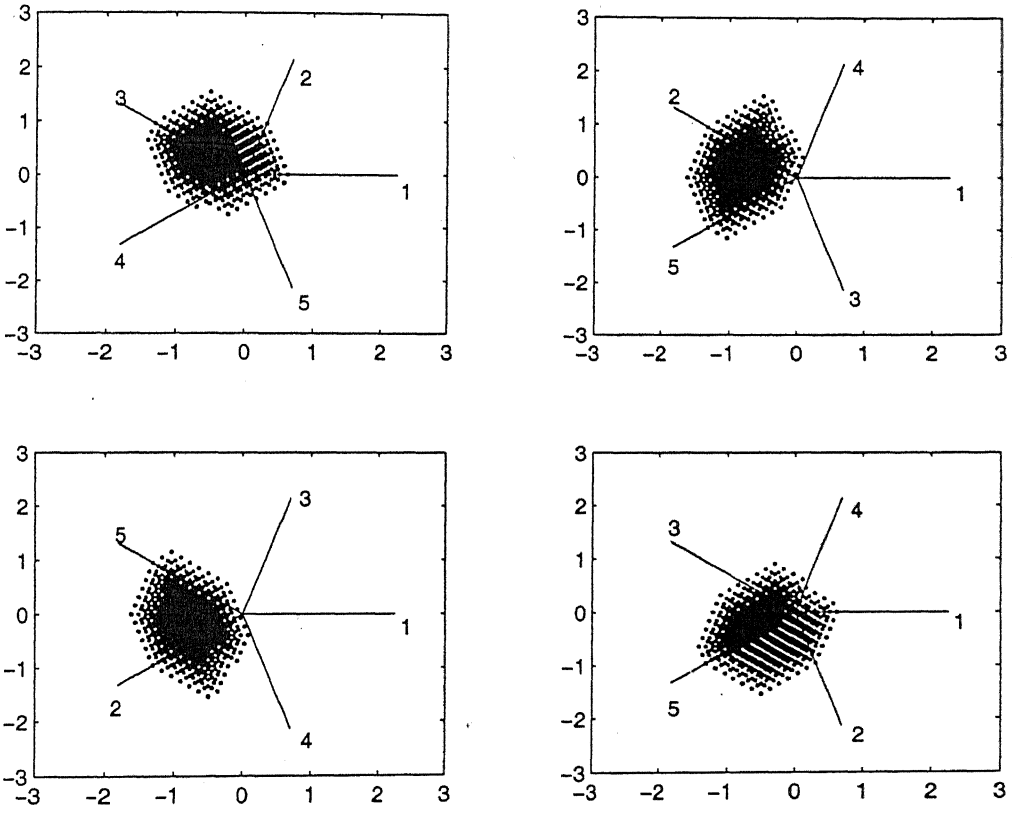


Figure 5.12: **5-DOF Workcell 1 C-obstacle 3**

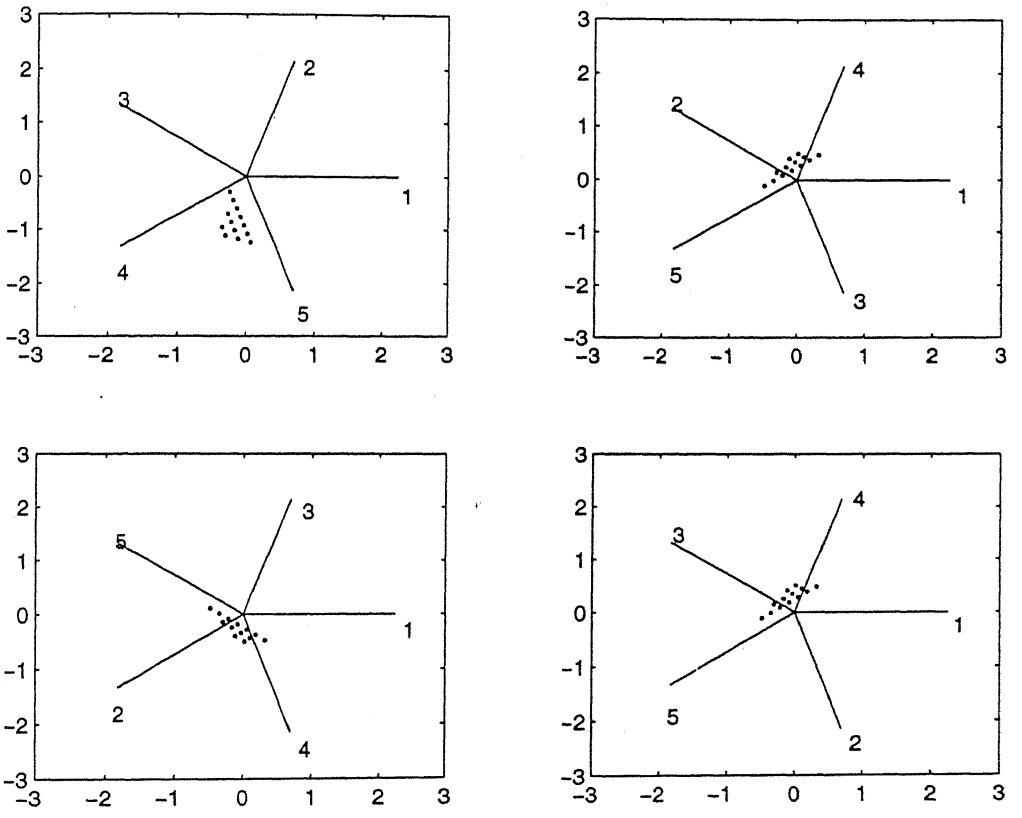


Figure 5.13: **5-DOF Workcell 1 C-obstacle 4**

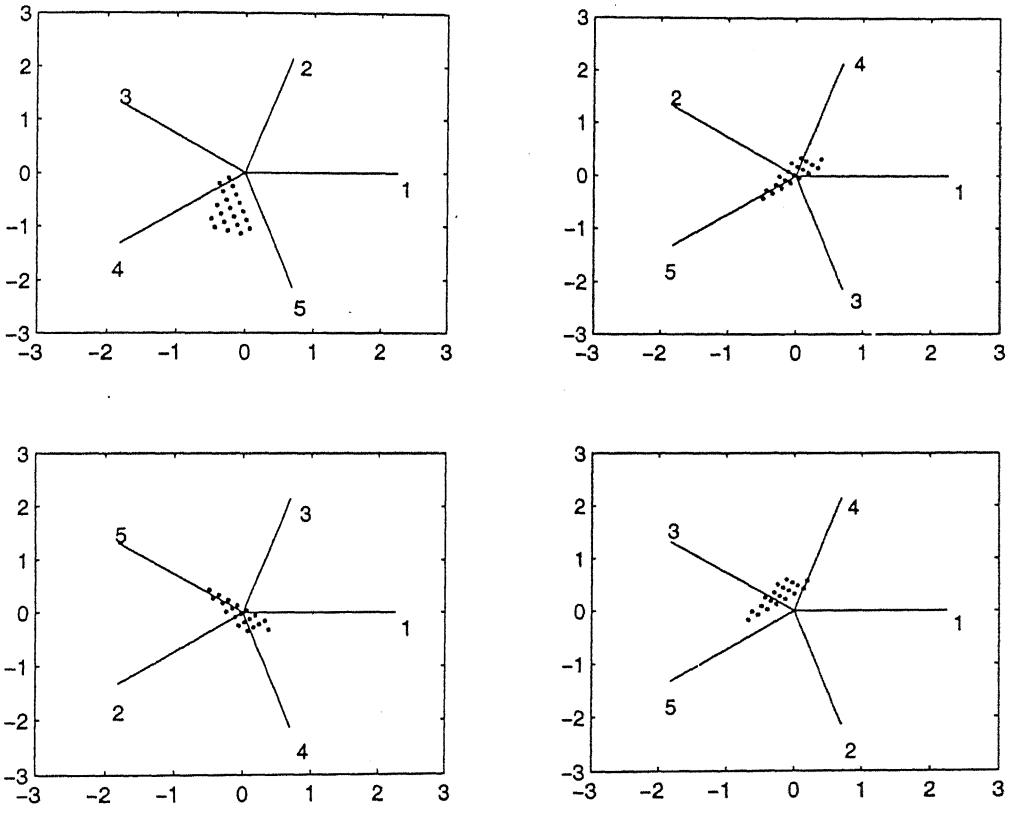


Figure 5.14: **5-DOF Workcell 1 C-obstacle 5**

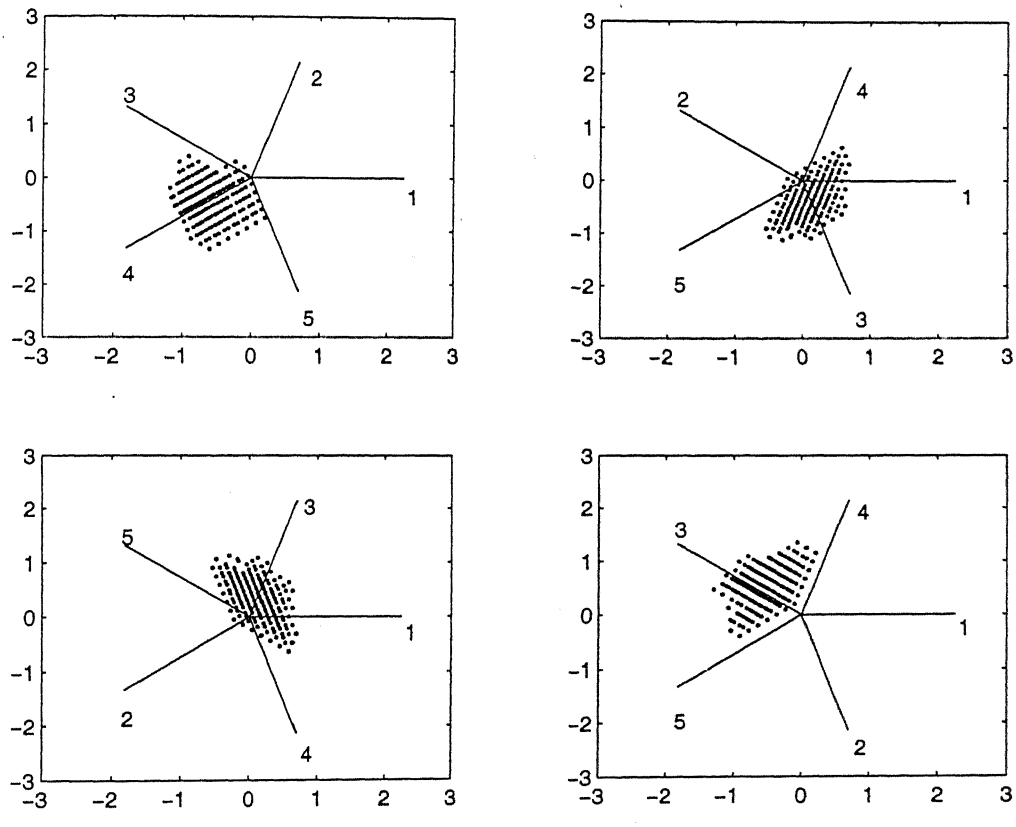


Figure 5.15: **5-DOF Workcell 1 C-obstacle 6**

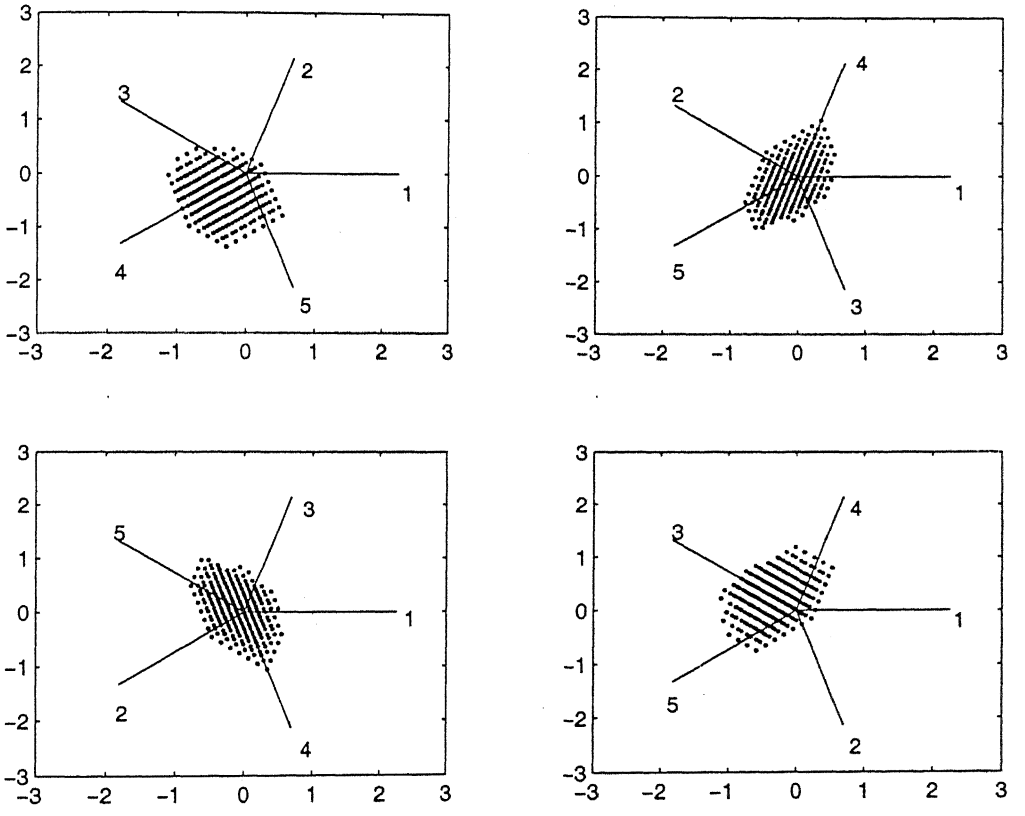


Figure 5.16: **5-DOF Workcell 1 C-obstacle 7**

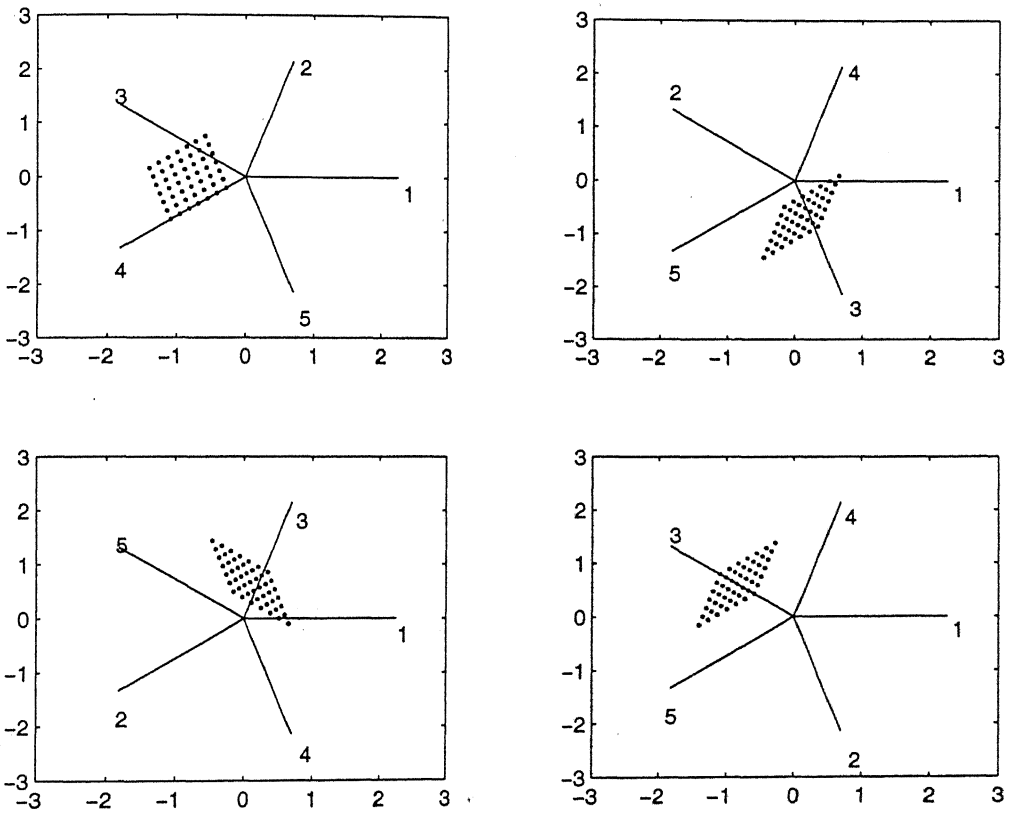


Figure 5.17: **5-DOF Workcell 1 C-obstacle 8**

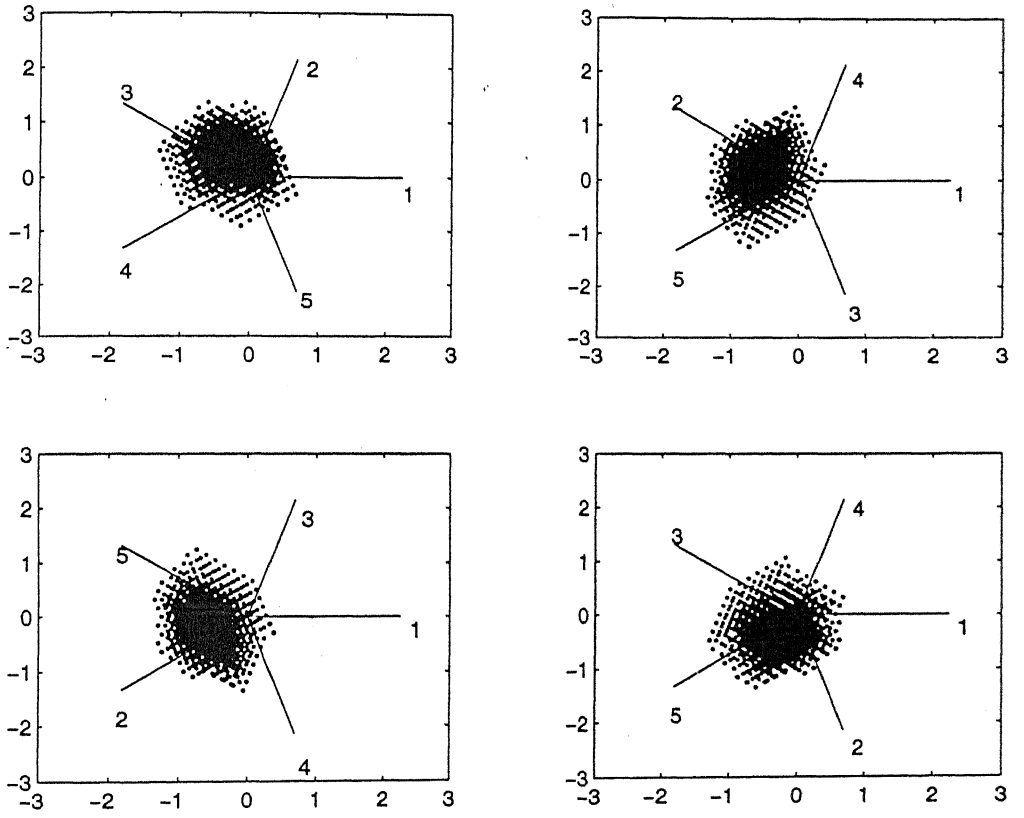


Figure 5.18: **5-DOF Workcell 1 C-obstacle 9**

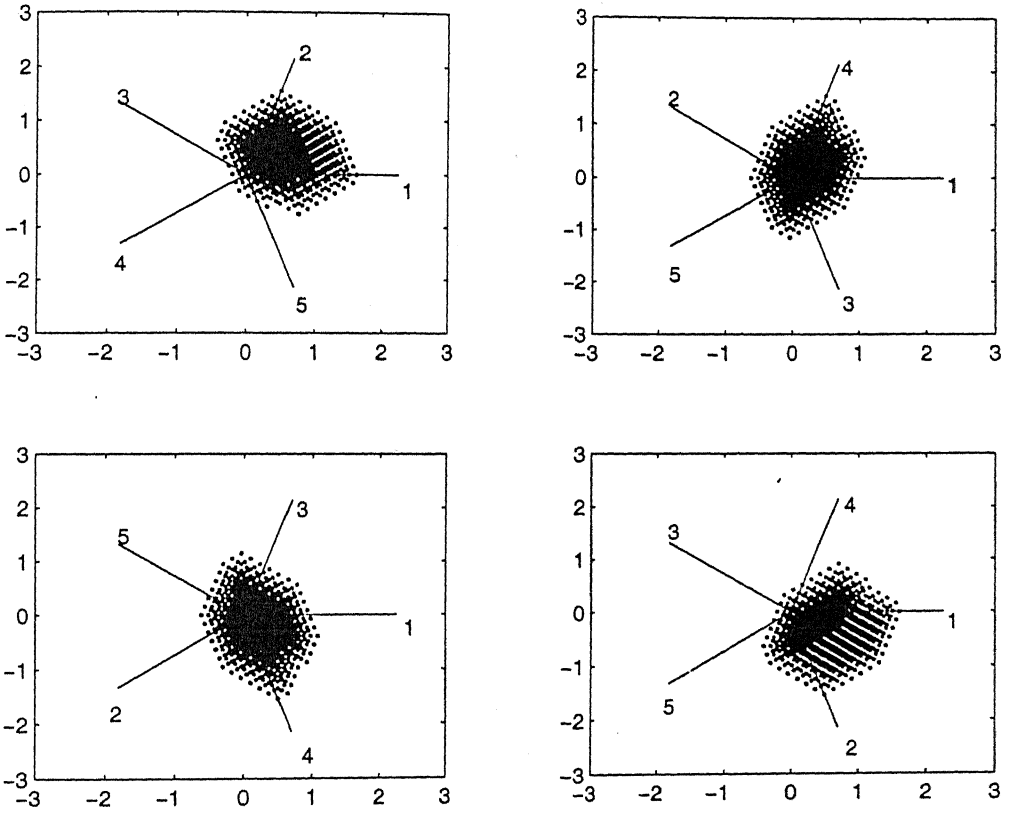


Figure 5.19: **5-DOF Workcell 1 C-obstacle 10**

The plots depicting C-obstacles have been shown for four different permutations of axes' location. This has been done to convey contiguity information regarding the C-obstacles. It may happen that non-contiguous points are reported as contiguous in one projection, but if they are shown as contiguous in all four projections, it is very likely that the C-obstacle is actually contiguous.

From the shown plots it is evident that all obtained C-obstacles are connected, i.e. points belonging to a (reported) C-obstacle are indeed contiguous. It might be noticed that some of the C-obstacles are very small, consisting of a small number of sampled points. These are mostly C-obstacles resulting out of self-intersection of the manipulator and , though small, offer significant hindrance to path planning.

Workcell 2

Dimensions (length, height, breadth) : 200 200 200
Resolution Factor : 1.0

Number of obstacles : 2

For obstacle 1
Frame origin co-ordinates wrt {U} frame : 100 60 100
Frame orientation wrt {U} frame (RPY angles) : 0 0 0
Number of primitives : 3
For primitive 1
Type : parallelopiped
Length : 10
Height : 10
Breadth : 10
Frame origin co-ordinates wrt obstacle frame : 0 0 0
Frame orientation wrt obstacle frame (RPY angles) : 0 0 0
For primitive 2
Type : cylindrical
Inner Radius : 0
Outer Radius : 5
Start Angle : 0
End Angle : 360
Length : 10
Frame origin co-ordinates wrt obstacle frame : 5 10 5
Frame orientation wrt obstacle frame (RPY angles) : 0 0 0
For primitive 3
Type : parallelopiped
Length : 10
Height : 10
Breadth : 10
Frame origin co-ordinates wrt obstacle frame : 0 20 0
Frame orientation wrt obstacle frame (RPY angles) : 0 0 0

For obstacle 2

Frame origin co-ordinates wrt {U} frame : 140 70 110

Frame orientation wrt {U} frame (RPY angles) : 0 0 0

Number of primitives : 1

For primitive 1

Type : cylindrical

Inner Radius : 5

Outer Radius : 10

Start Angle : 10

End Angle : 350

Length : 20

Frame origin co-ordinates wrt obstacle frame : 0 0 0

Frame orientation wrt obstacle frame (RPY angles) : 0 0 0

Postion and orientation of manipulator

Frame origin co-ordinates wrt {U} frame : 120 40 110

Frame orientation wrt {U} frame (RPY angles) : 0 0 90

Number of steps between limits of joint variables : 6

The given workcell is roughly drawn in Fig 5.17

Total number of C-obstacles detected = 79

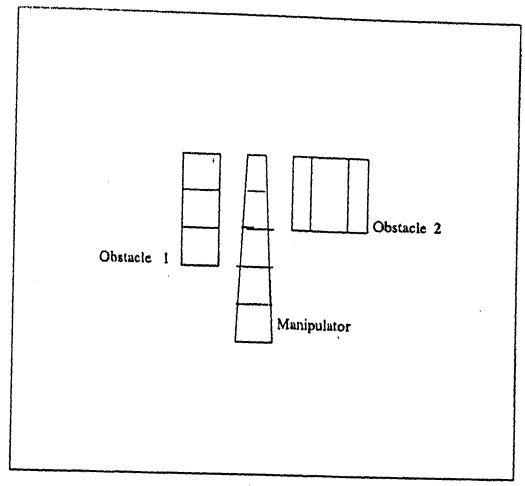


Figure 5.20: **Workcell 2 (5-DOF, Planar)**

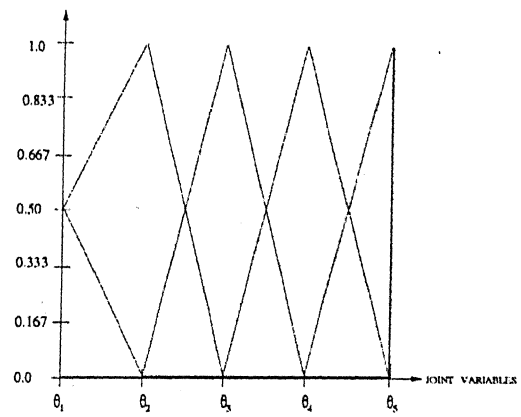


Figure 5.21: **5-DOF Workcell 2 Boundary Configurations**

Here, from the plot for boundary configurations, it can be seen that for given limits on the joint variables, the manipulator hits the workcell boundary for a very few number of configurations and those configurations relate to the limiting values of θ's. Also, the plot for obstacle 2 in this case, is a sparse one indicating that the manipulator does not hit it too often.

From the plots for C-obstacles (10 of them are reported here), it can be seen that all of them are contiguous ones.

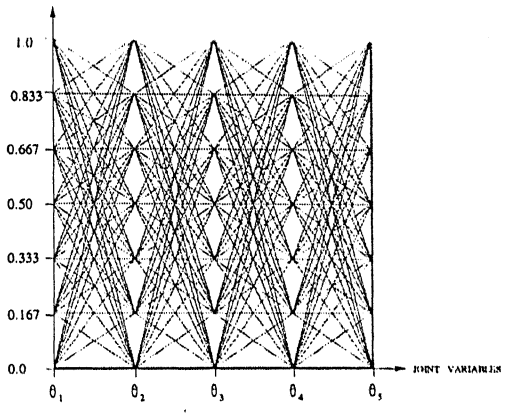


Figure 5.22: **5-DOF Workcell 2 Self-intersection Configurations**

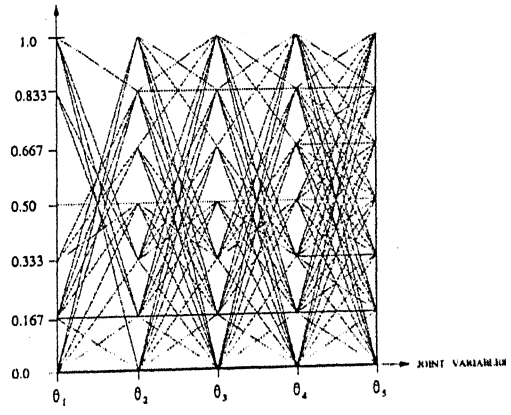


Figure 5.23: **5-DOF Workcell 2 Obstacle 1 Configurations**

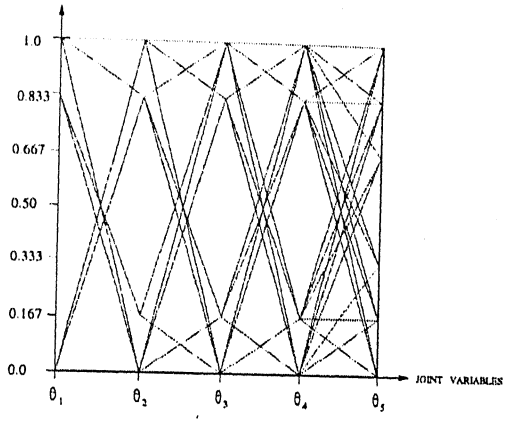


Figure 5.24: **5-DOF Workcell 2 Obstacle 2 Configurations**

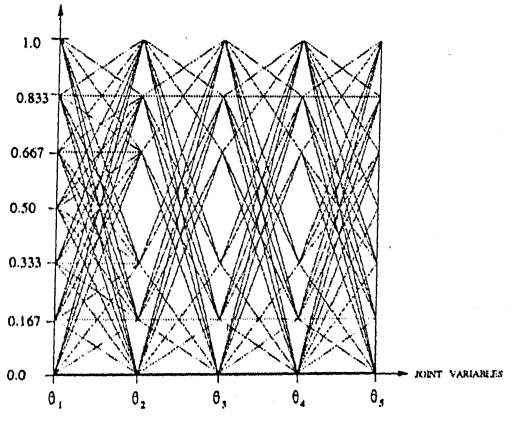


Figure 5.25: **5-DOF Workcell 2 Free Configurations**

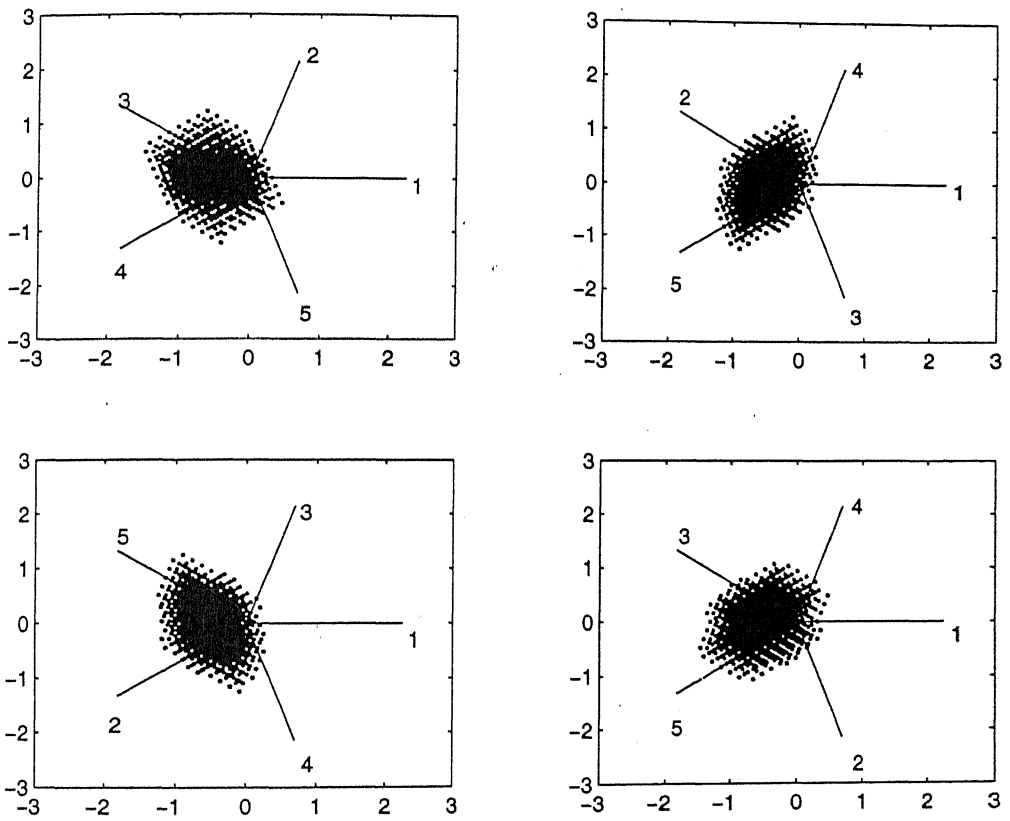


Figure 5.26: **5-DOF Workcell 2 C-obstacle 1**

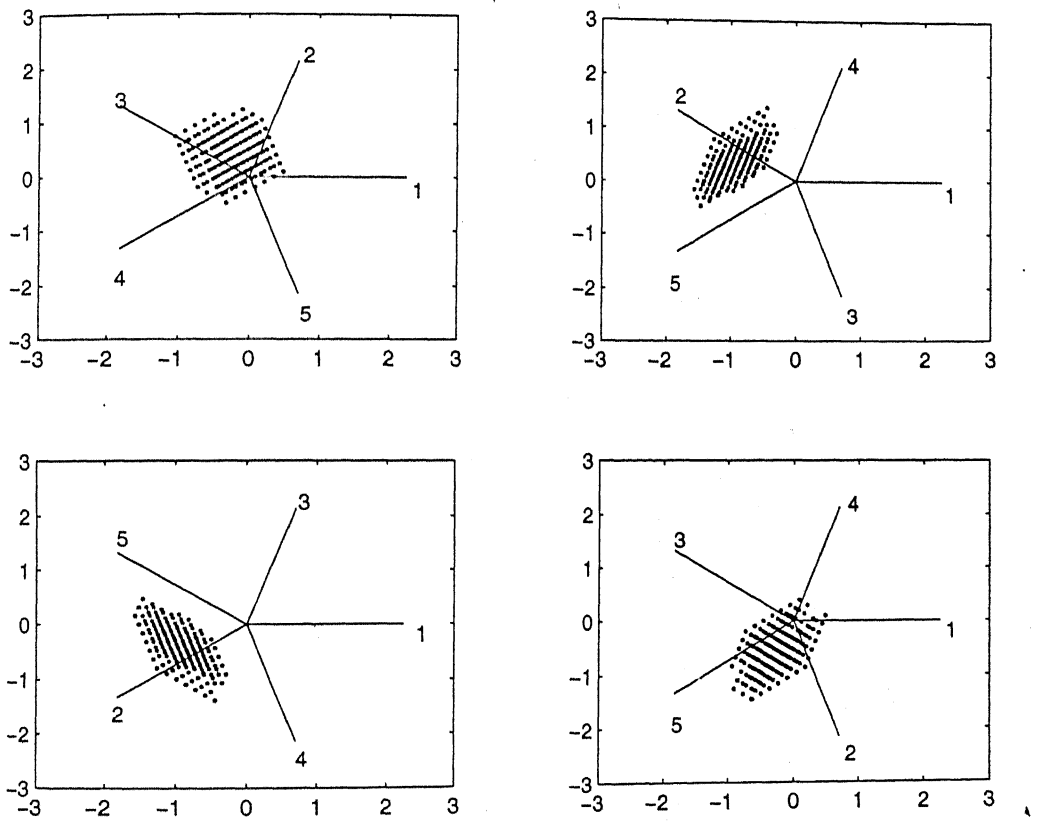


Figure 5.27: **5-DOF Workcell 2 C-obstacle 2**

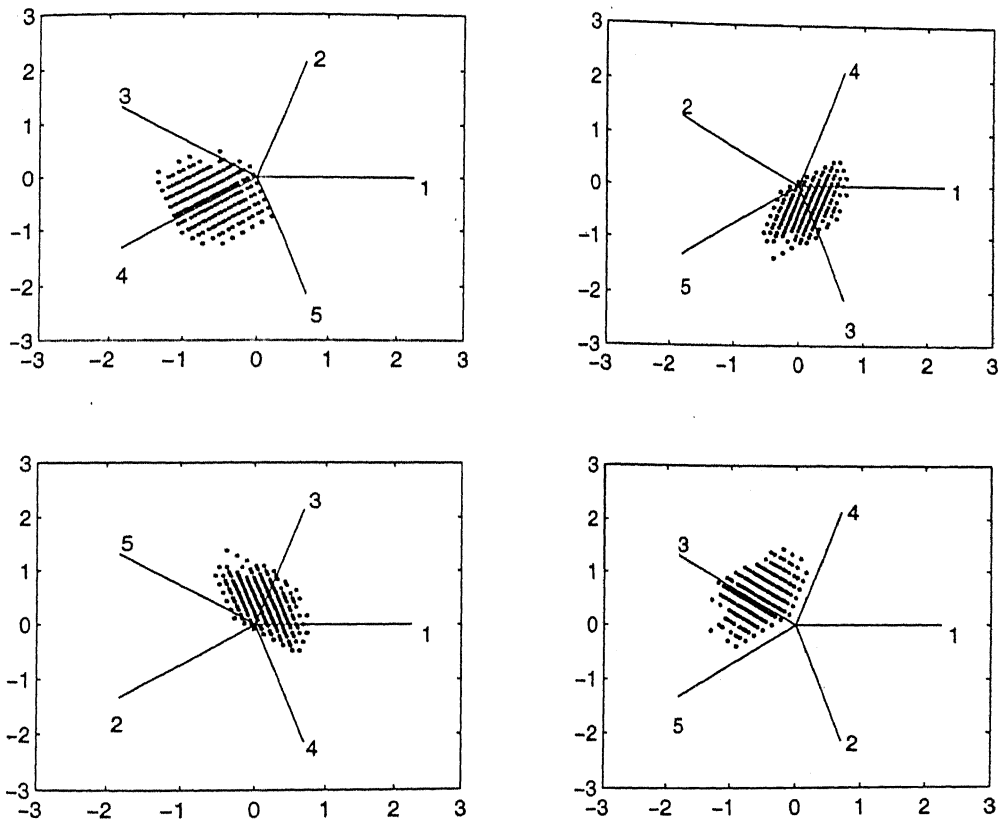


Figure 5.28: **5-DOF Workcell 2 C-obstacle 3**

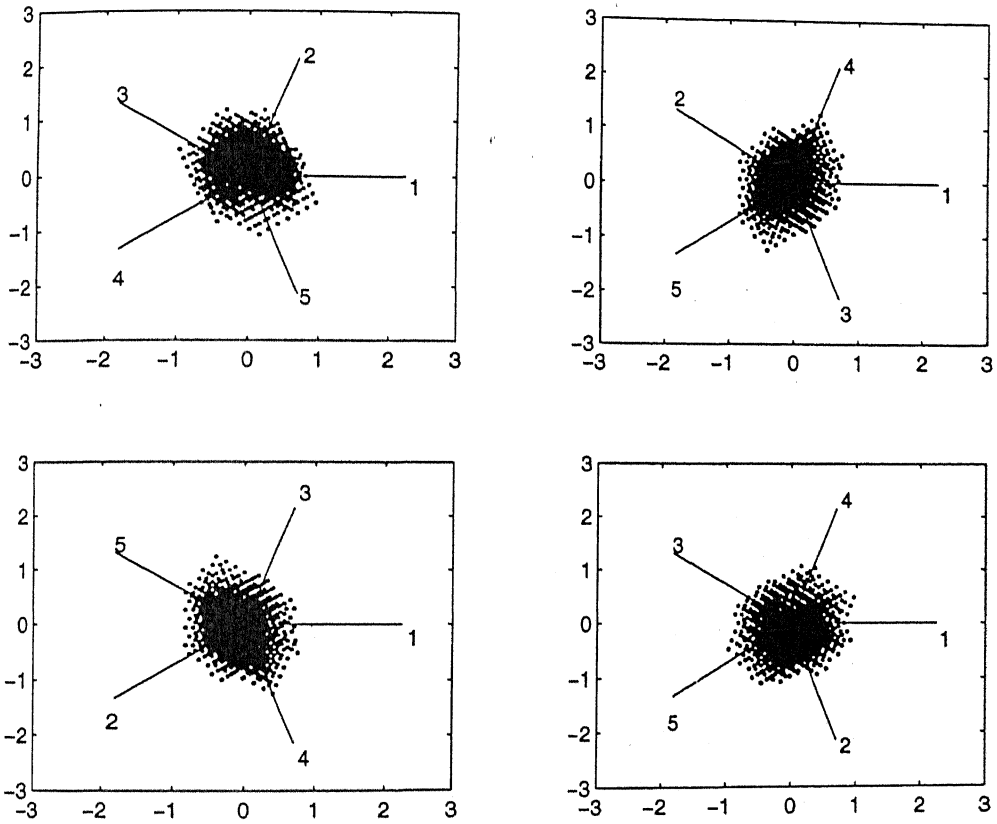


Figure 5.30: **5-DOF Workcell 2 C-obstacle 5**

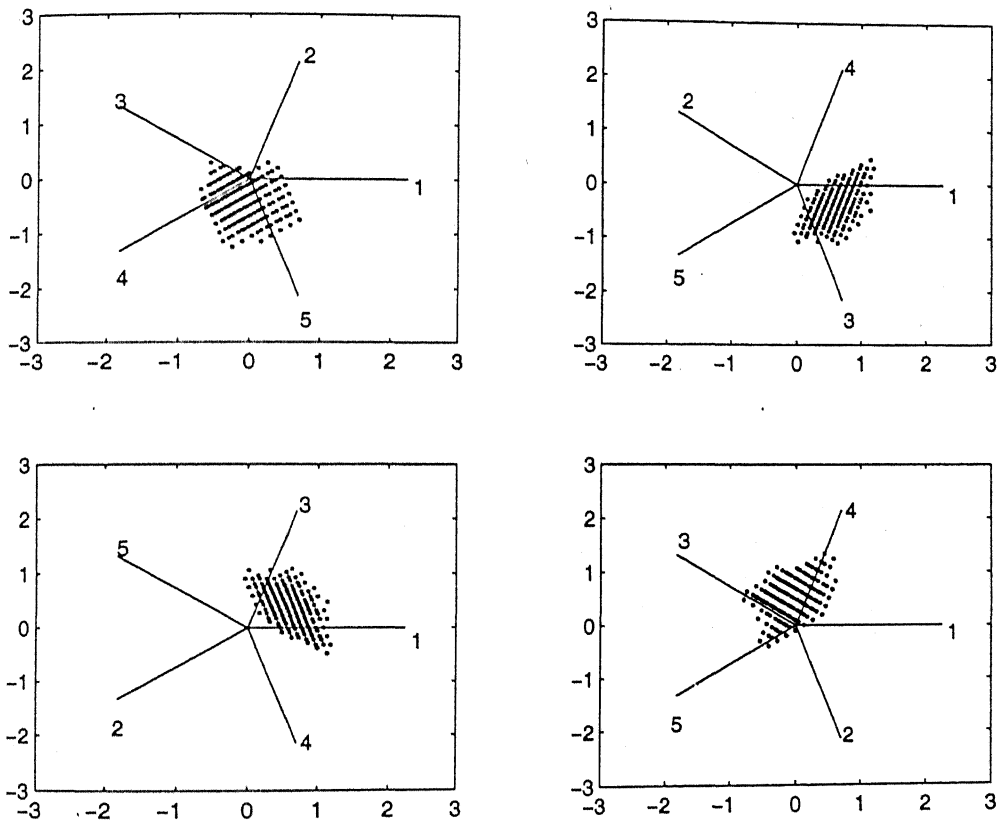


Figure 5.31: **5-DOF Workcell 2 C-obstacle 6**

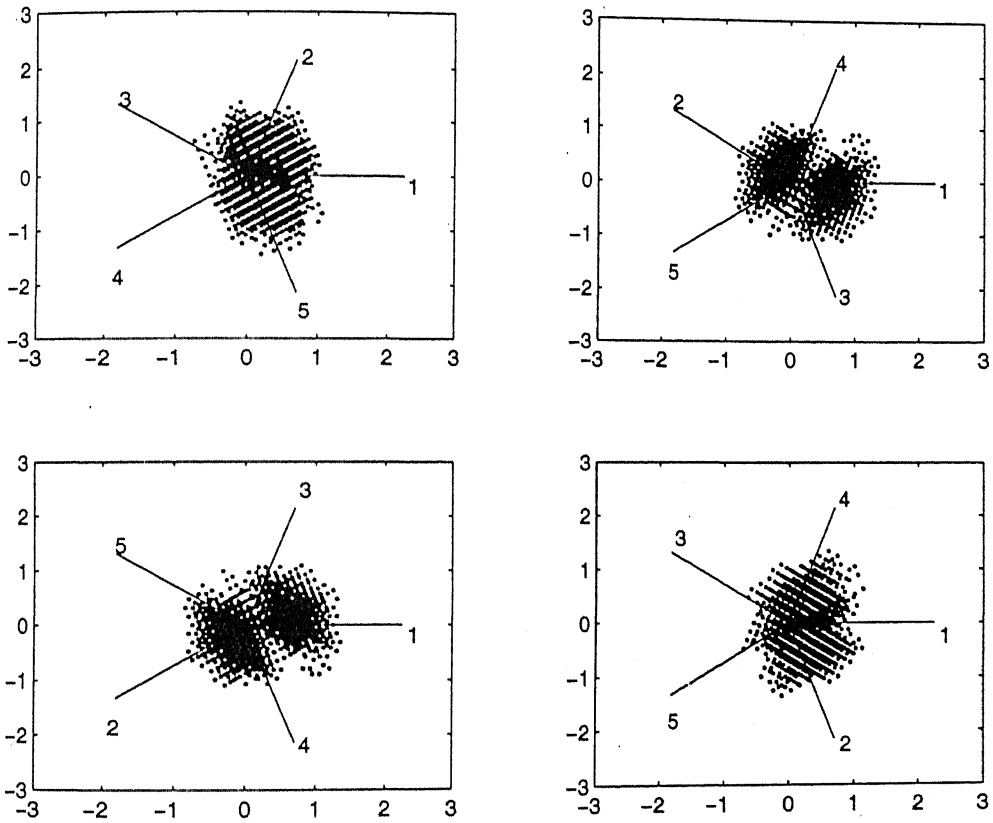


Figure 5.32: **5-DOF Workcell 2 C-obstacle 7**

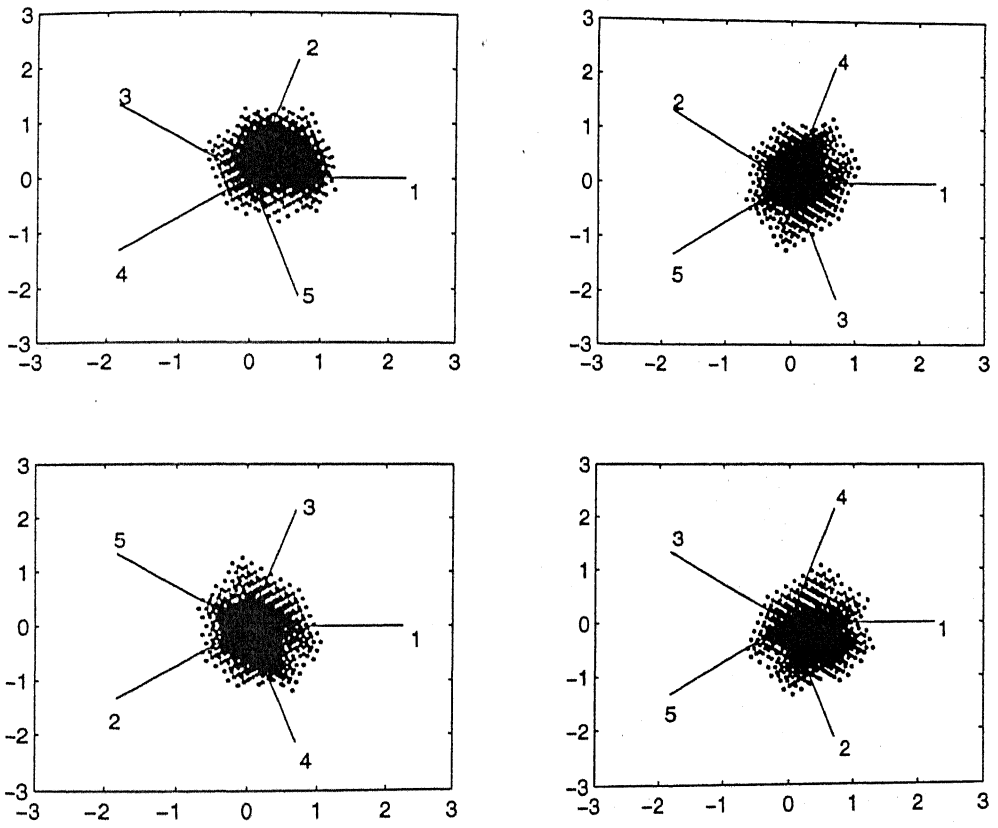


Figure 5.33: **5-DOF Workcell 2 C-obstacle 8**

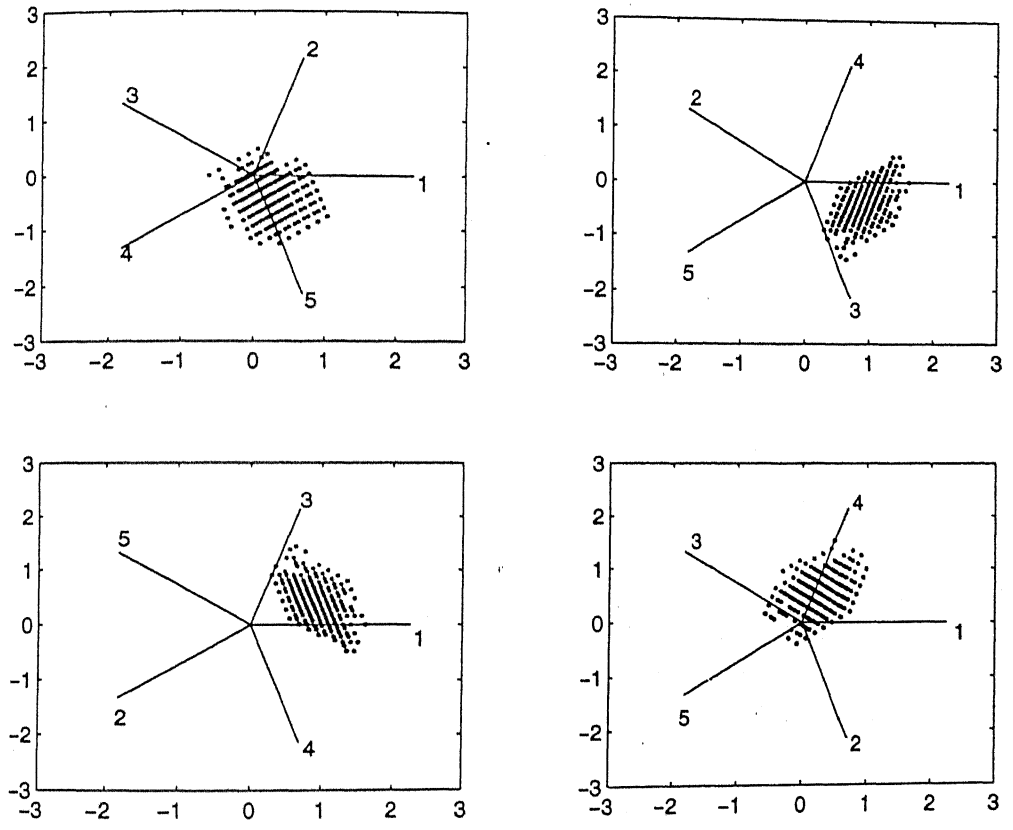


Figure 5.34: **5-DOF Workcell 2 C-obstacle 9**

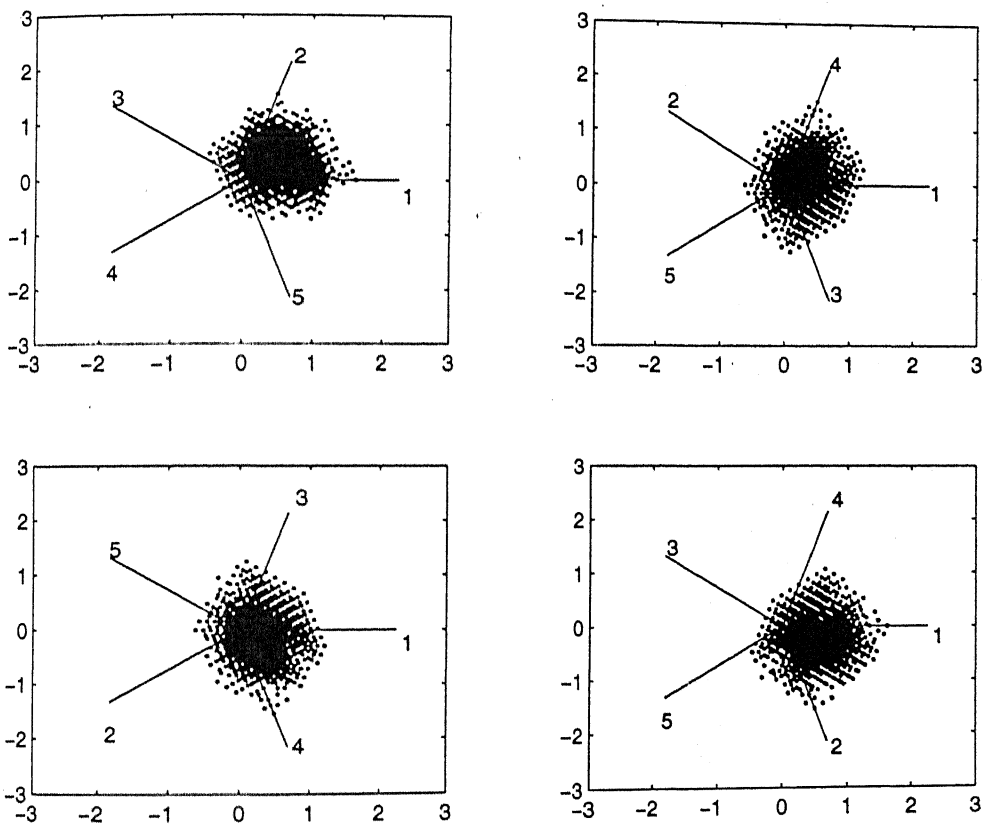


Figure 5.35: **5-DOF Workcell 2 C-obstacle 10**

5.2.2 Spatial 8-DOF hyper-redundant manipulator

Description of manipulator

Joint	Type	α_{i-1}	a_{i-1}	d_i	θ_i	Length	Limits
1	Revolute	0	0	0	θ_1	24.0	0 360
2	Revolute	-90	0	0	θ_2	22.0	-225 45
3	Revolute	0	22.0	0	θ_3	20.0	-135 135
4	Prismatic	0	20.0	d_4	0	18.0	-8 0
5	Prismatic	0	0	d_5	-90	16.0	29 34
6	Revolute	-90	0	0	θ_6	14.0	-180 0
7	Revolute	0	14.0	0	θ_7	12.0	-90 90
8	Prismatic	90	12	d_8	0	10.0	-5 0

Base radius of base link : 5.0

Radius reduction ratio : 0.997

Workcell 1

Dimensions (length, height, breadth) : 200 200 200

Resolution Factor : 1.0

Number of obstacles : 3

For obstacle 1

Frame origin co-ordinates wrt {U} frame : 80 40 70

Frame orientation wrt {U} frame (RPY angles) : 0 90 0

Number of primitives : 3

For primitive 1

Type : parallelopiped

Length : 15

Height : 5

Breadth : 5

Frame origin co-ordinates wrt obstacle frame : 0 0 0

Frame orientation wrt obstacle frame (RPY angles) : 0 0 0

For primitive 2

Type : parallelopiped

Length : 10
Height : 5
Breadth : 5
Frame origin co-ordinates wrt obstacle frame : 0 5 0
Frame orientation wrt obstacle frame (RPY angles) : 0 0 0
For primitive 3
Type : parallelopiped
Length : 5
Height : 5
Breadth : 5
Frame origin co-ordinates wrt obstacle frame : 0 10 0
Frame orientation wrt obstacle frame (RPY angles) : 0 45 0

For obstacle 2
Frame origin co-ordinates wrt {U} frame : 100 10 30
Frame orientation wrt {U} frame (RPY angles) : 90 0 90
Number of primitives : 2
For primitive 1
Type : cylindrical
Inner Radius : 0
Outer Radius : 10
Start Angle : 0
End Angle : 360
Length : 20
Frame origin co-ordinates wrt obstacle frame : 10 0 10
Frame orientation wrt obstacle frame (RPY angles) : 0 0 0
For primitive 2
Type : cylindrical
Inner Radius : 3
Outer Radius : 6
Start Angle : 10
End Angle : 200
Length : 15

Frame origin co-ordinates wrt obstacle frame : 10 20 10
Frame orientation wrt obstacle frame (RPY angles) : 0 0 0

For obstacle 3
Frame origin co-ordinates wrt {U} frame : 150 40 70
Frame orientation wrt {U} frame (RPY angles) : 0 0 45
Number of primitives : 2
For primitive 1
Type : parallelopiped
Length : 20
Height : 10
Breadth : 10
Frame origin co-ordinates wrt obstacle frame : 0 0 0
Frame orientation wrt obstacle frame (RPY angles) : 0 0 90
For primitive 2
Type : cylindrical
Inner Radius : 8
Outer Radius : 10
Start Angle : 0
End Angle : 360
Length : 20
Frame origin co-ordinates wrt obstacle frame : 5 20 5
Frame orientation wrt obstacle frame (RPY angles) : 0 0 -90

Postion and orientation of manipulator
Frame origin co-ordinates wrt {U} frame : 80 30 30
Frame orientation wrt {U} frame (RPY angles) : -90 180 0

Number of steps between limits of joint variables : 4

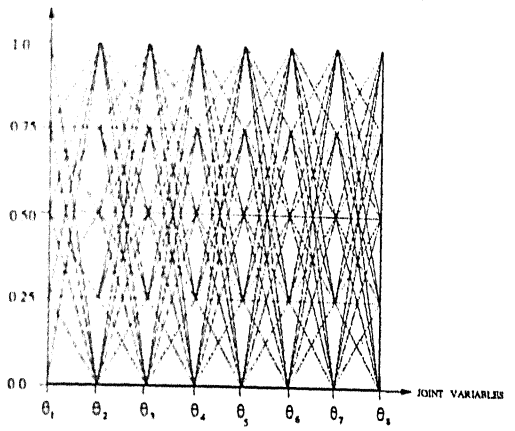


Figure 5.36: **8-DOF Workcell 1 Boundary Configurations**

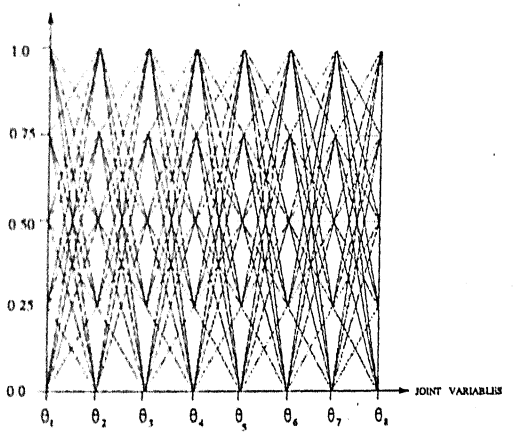


Figure 5.37: **8-DOF Workcell 1 Self-intersection Configurations**

From the plots it can be made out that for most of the values of θ_1, θ_2 and θ_3 the manipulator does not hit any obstacle. Also, a large number of free configurations are available.

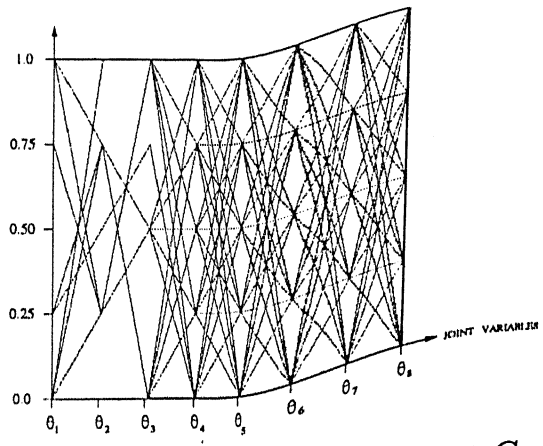
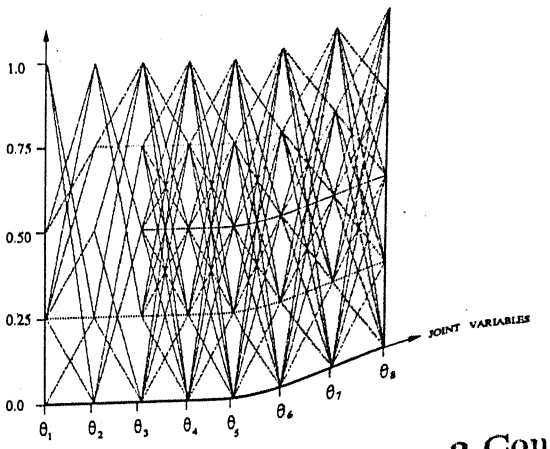


Figure 5.38: 8-DOF Workcell 1 *Obstacle* 1 Configurations



'igure 5.39: 8-DOF Workcell 1 *Obstacle* 2 Configurations

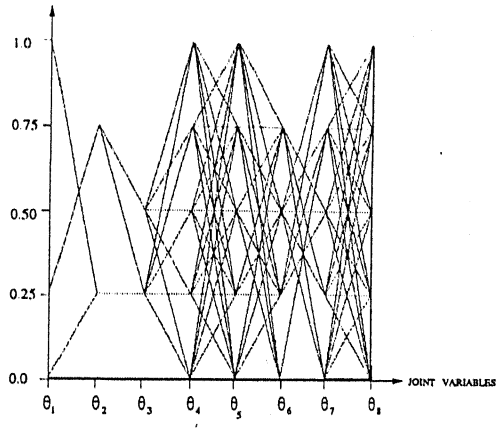


Figure 5.40: **8-DOF Workcell 1 Obstacle 3 Configurations**

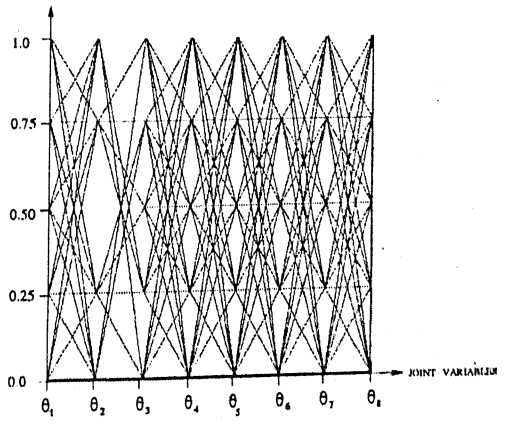


Figure 5.41: **8-DOF Workcell 1 Free Configurations**

Workcell 2

Dimensions (length, height, breadth) : 200 200 200
Resolution Factor : 1.0

Number of obstacles : 2

For obstacle 1
Frame origin co-ordinates wrt {U} frame : 50 70 30
Frame orientation wrt {U} frame (RPY angles) : 0 90 0
Number of primitives : 2
For primitive 1
Type : parallelopiped
Length : 15
Height : 5
Breadth : 5
Frame origin co-ordinates wrt obstacle frame : 0 0 0
Frame orientation wrt obstacle frame (RPY angles) : 0 0 0
For primitive 2
Type : parallelopiped
Length : 10
Height : 5
Breadth : 5
Frame origin co-ordinates wrt obstacle frame : 0 5 0
Frame orientation wrt obstacle frame (RPY angles) : 0 0 0

For obstacle 2
Frame origin co-ordinates wrt {U} frame : 130 50 60
Frame orientation wrt {U} frame (RPY angles) : 0 0 0
Number of primitives : 2
For primitive 1
Type : cylindrical
Inner Radius : 0
Outer Radius : 10

Start Angle : 0
End Angle : 360
Length : 10
Frame origin co-ordinates wrt obstacle frame : 10 0 10
Frame orientation wrt obstacle frame (RPY angles) : 0 0 0
For primitive 2
Type : cylindrical
Inner Radius : 0
Outer Radius : 6
Start Angle : 10
End Angle : 200
Length : 5
Frame origin co-ordinates wrt obstacle frame : 10 10 10
Frame orientation wrt obstacle frame (RPY angles) : 0 0 0

Postion and orientation of manipulator
Frame origin co-ordinates wrt {U} frame : 80 30 30
Frame orientation wrt {U} frame (RPY angles) : -90 180 0

Number of steps between limits of joint variables : 4

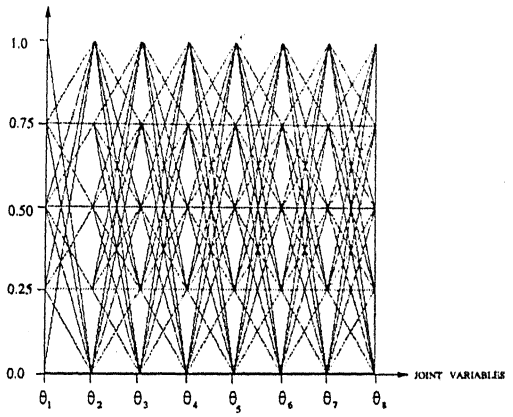


Figure 5.42: **8-DOF Workcell 2 Boundary Configurations**

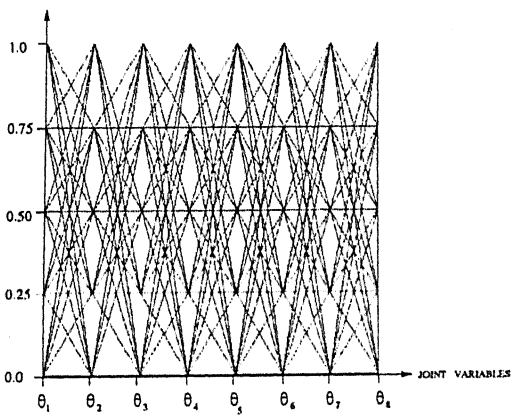


Figure 5.43: **8-DOF Workcell 2 Self-Intersection Configurations**

The plots show that for most values of θ_1, θ_2 and θ_3, the manipulator does not hit any obstacle. There are a lot of configurations in which the manipulator hits itself or the workcell boundary. A large number of free configurations are also shown.

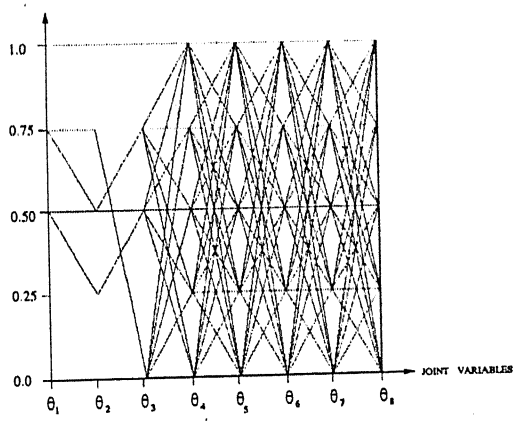


Figure 5.44: **8-DOF Workcell 2 Obstacle 1 Configurations**

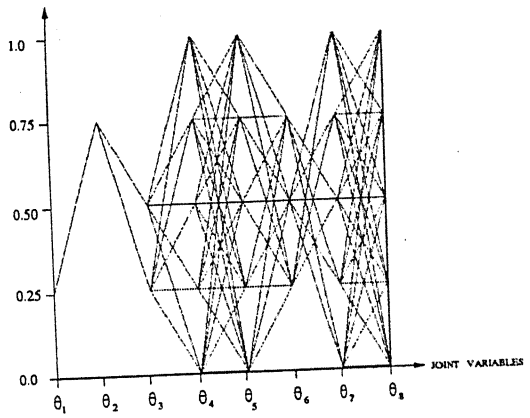


Figure 5.45: **8-DOF Workcell 2 Obstacle 2 Configurations**

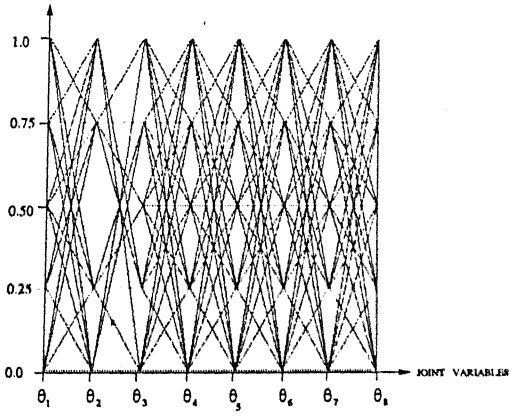


Figure 5.46: **8-DOF Workcell 2 Free Configurations**

5.3 Inverse Kinematic Solutions

Inverse kinematics has been performed for the same 5-DOF planar and 8-DOF spatial hyper-redundant manipulators. For each of them, results for two sets of end-effector poses are reported here.

5.3.1 Planar 5-DOF hyper-redundant manipulator

Joint	Type	α_{i-1}	a_{i-1}	d_i	θ_i	Length	Limits
1	Revolute	0	0	0	θ_1	8.0	10 350
2	Revolute	0	8.0	0	θ_2	8.0	10 350
3	Revolute	0	8.0	0	θ_3	8.0	10 350
4	Revolute	0	8.0	0	θ_4	8.0	10 350
5	Revolute	0	8.0	0	θ_5	8.0	10 350

Case 1

Population size : 300
Number of generations : 500
Distribution index for crossover : 25
Distribution index for mutation : 100
Crossover probability : 0.9
Mutation probability : 0.009
Random seed : 0.7654321
Position and orientation of manipulator base frame
Frame origin co-ordinates wrt {U} frame : 100 10 100
Frame orientation wrt {U} frame (RPY angles) : 0 0 0
Position and orientation of end-effector frame
Frame origin co-ordinates wrt {U} frame : 100 60 100
Frame orientation wrt {U} frame (RPY angles) : 0 0 0

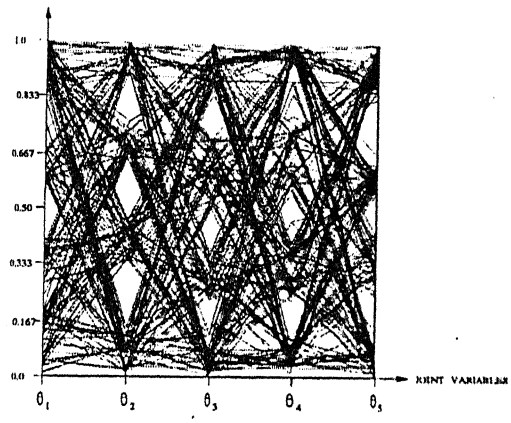


Figure 5.47: **Solution Clusters for 5-DOF Case 1**

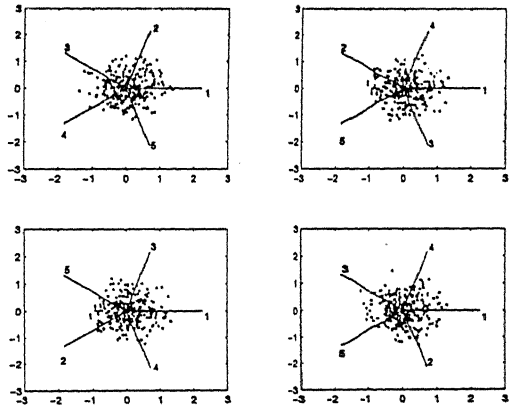


Figure 5.48: **Solution Clusters for 5-DOF Case 1**

Case 2

Population size : 300
Number of generations : 500
Distribution index for crossover : 25
Distribution index for mutation : 100
Crossover probability : 0.9
Mutation probability : 0.009
Random seed : 0.7654321
Postion and orientation of manipulator base frame
Frame origin co-ordinates wrt {U} frame : 100 10 100
Frame orientation wrt {U} frame (RPY angles) : 0 0 0
Postion and orientation of end-effector frame
Frame origin co-ordinates wrt {U} frame : 80 50 100
Frame orientation wrt {U} frame (RPY angles) : 0 0 0

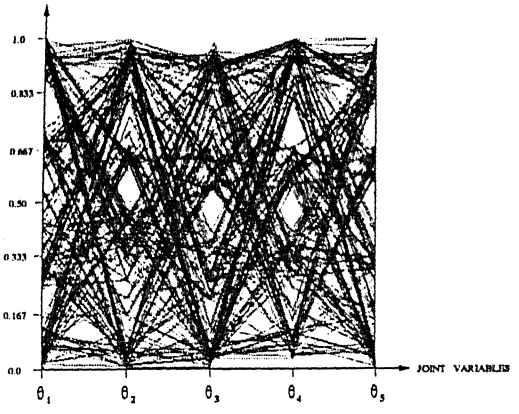


Figure 5.49: **Solution Clusters for 5-DOF Case 2**

From the plots it can be made out that the solutions tend to crowd around some values of θ's. These are the solution 'clusters' discussed before. Some of the stray solutions are present because of the mutation operator used in GA.

The solutions provided here are for workcells which do not have any obstacles. For a particular pose of the end-effector, this gives us C-space points

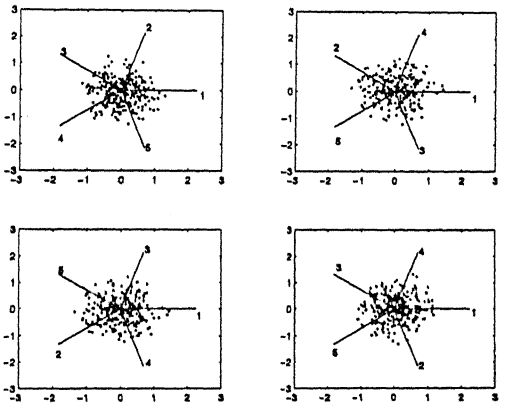


Figure 5.50: **Solution Clusters for 5-DOF Case 2**

in the inverse kinematics solutions' region. When C-obstacles are present, the solution cluster may lie wholly outside, wholly inside or partly inside the C-obstacle. This gives us information about the feasibility of the obtained inverse kinematic solutions.

5.3.2 Spatial 8-DOF hyper-redundant manipulator

Joint	Type	α_{i-1}	a_{i-1}	d_i	θ_i	Length	Limits
1	Revolute	0	0	0	θ_1	24.0	0 360
2	Revolute	-90	0	0	θ_2	22.0	-225 45
3	Revolute	0	22.0	0	θ_3	20.0	-135 135
4	Prismatic	0	20.0	d_4	0	18.0	-8 0
5	Prismatic	0	0	d_5	-90	16.0	29 34
6	Revolute	-90	0	0	θ_6	14.0	-180 0
7	Revolute	0	14.0	0	θ_7	12.0	-90 90
8	Prismatic	90	12	d_8	0	10.0	-5 0

Case 1

Population size : 300

Number of generations : 500

Distribution index for crossover : 25

Distribution index for mutation : 100

Crossover probability : 0.9

Mutation probability : 0.009

Random seed : 0.7654321

Postion and orientation of manipulator base frame

Frame origin co-ordinates wrt {U} frame : 80 30 30

Frame orientation wrt {U} frame (RPY angles) : -90 180 0

Position and orientation of end-effector frame

Frame origin co-ordinates wrt {U} frame : 80 50 60

Frame orientation wrt {U} frame (RPY angles) : 0 0 0

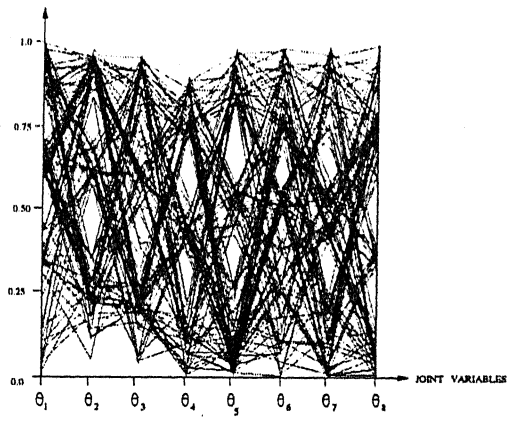


Figure 5.51: **Solution Clusters for 8-DOF Case 1**

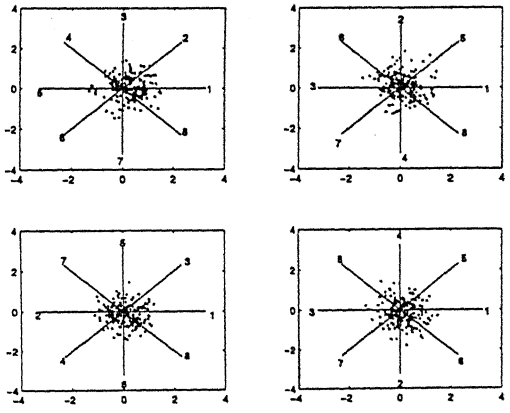


Figure 5.52: **Solution Clusters for 8-DOF Case 1**

Case 2

Population size : 300

Number of generations : 500

Distribution index for crossover : 25

Distribution index for mutation : 100

Crossover probability : 0.9

Mutation probability : 0.009

Random seed : 0.7654321

Postion and orientation of manipulator base frame

Frame origin co-ordinates wrt {U} frame : 80 30 30

Frame orientation wrt {U} frame (RPY angles) : -90 180 0

Postion and orientation of end-effector frame

Frame origin co-ordinates wrt {U} frame : 120 60 30

Frame orientation wrt {U} frame (RPY angles) : -90 0 0

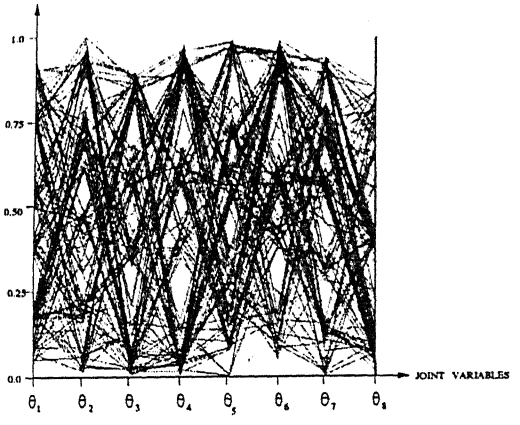


Figure 5.53: **Solution Clusters for 8-DOF Case 2**

Here also, the solutions tend to crowd around some values of θ's as expected.

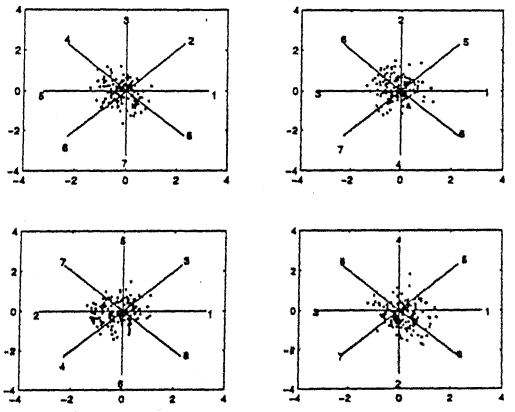


Figure 5.54: **Solution Clusters for 8-DOF Case 2**

Chapter 6

Conclusions and Future Scope

6.1 Summary

In the present work, attempt has been made to get the C-space representation of hyper-redundant manipulators for 2-D and 3-D cases. The obtained C-space points are n-dimensional ($n > 3$). But since there is no available technique for representing n-dimensional points, the results have been presented as 2-D graphs. Though not self-explanatory, they do convey information about the various types of configurations in the C-space. The contiguity of different C-obstacles can be seen in the projection type plots. The mapping that we have obtained in the various cases illustrated before is reasonably exhaustive if not complete. The density of points can be further increased by increasing the number of 'steps' though it becomes computationally intensive. For presentation purposes, low values of 'steps' have been used.

While getting C-obstacles, it is important to avoid overlapping of different C-obstacles. The same point should not be included in two separate C-obstacles. The representation of C-obstacles presented in this work is not complete, but it does solve the problem of overlapping. Instead of complete C-obstacles, we may get fragments which belong to the same C-obstacle.

The process of computing inverse kinematic solutions presented in this work is simple to implement. As expected, we get solution clusters pertaining to different peaks in the function space. These solutions are good solutions

as long as the goal point lies inside the workspace of the manipulator. Points lying outside the workspace, can be detected by the very poor fitness of the solutions.

6.2 Future Scope

The present work opens up quite a few avenues for future continuations. It is one of the most important components for implementing path planning schemes which are based on C-space. For such planning schemes, a good description of the C-space is required, which this work provides. The inverse kinematics solutions can be used to get tentative solutions and they can be compared with the obtained description of the C-space to check for feasibility of obtained paths.

The C-space description is roughly shown in Fig 6.1. In this figure, it can be seen that some solution sets lie wholly inside a C-obstacle, some lie wholly outside and some partly outside. The comparison between the two can be quantified by computing the 'distance' between the C-obstacle points and the inverse kinematics solution points. Minimizing the interaction of C-obstacles and solution clusters, the designer can synthesize better manipulators.Thus this work provides the necessary foundation for path planning and synthesis.

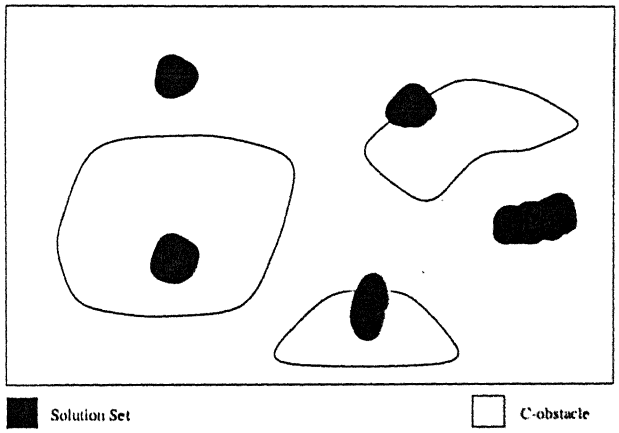


Figure 6.1: C-space

Appendix A

Configuration Space

A.1 Configuration Space

The Configuration space (C-space) of a manipulator is an n-dimensional space where each dimension represents one degree of freedom of the manipulator. For example, let us consider a manipulator having 10 joints, i.e. it has 10 degrees of freedom. The C-space for this manipulator will be a 10-dimensional space where each dimension (axis) corresponds to each degree of freedom of the manipulator.

A point to note here is that a point in the C-space of a manipulator actually represents the (unique) configuration that the manipulator takes in the 3-dimensional Cartesian space. Conversely, every configuration of the manipulator in the 3-D Cartesian space maps to a point in its C-space.

A.2 Configuration

We consider a rigid object $\mathcal{A}$, the robot, moving in a physical workspace $\mathcal{W}$ (See Fig A.1). We represent $\mathcal{W}$ as the N-dimensional Euclidean space R^N, (here N = 3) equipped with a fixed Cartesian system or frame denoted by $\mathcal{F}_\mathcal{W}$. We represent $\mathcal{A}$ at a reference position and orientation as a compact subset of R^N. A moving frame $\mathcal{F}_\mathcal{A}$ is attached to $\mathcal{A}$ so that each point in the robot has fixed co-ordinates in $\mathcal{F}_\mathcal{A}$. The origins of $\mathcal{F}_\mathcal{W}$ and $\mathcal{F}_\mathcal{A}$ are denoted by $\mathcal{O}_\mathcal{W}$ and $\mathcal{O}_\mathcal{A}$ respectively. So a configuration $\mathbf{q}$ of $\mathcal{A}$ is a specification of the position and orientation of $\mathcal{F}_\mathcal{A}$ with respect to $\mathcal{F}_\mathcal{W}$. The *configuration*

space of $\mathcal{A}$ is the sapce $\mathcal{C}$ of all possible configurations of $\mathcal{A}$.

A configuration $\mathbf{q}$ of $\mathcal{A}$ can be bijectively represented, in a way that depends upon the choice of the frames $\mathcal{F}_\mathcal{A}$ and $\mathcal{F}_\mathcal{W}$, as a pair $(\mathcal{T}, \Theta)$, where $\mathcal{T}$ determines the position of the origin $\mathcal{O}_\mathcal{A}$ of $\mathcal{F}_\mathcal{A}$ in $\mathcal{F}_\mathcal{W}$ and Θ determines the orientation of $\mathcal{F}_\mathcal{A}$'s axis with respect to $\mathcal{F}_\mathcal{W}$.

Let us adopt the following convention: $\mathcal{T}$ is equal to N-vector of the coordinates of $\mathcal{O}_\mathcal{A}$ in $\mathcal{F}_\mathcal{W}$ and Θ is equal to $N \times N$ matrix whose columns are the components of the unit vector along $\mathcal{F}_\mathcal{A}$'s axis in

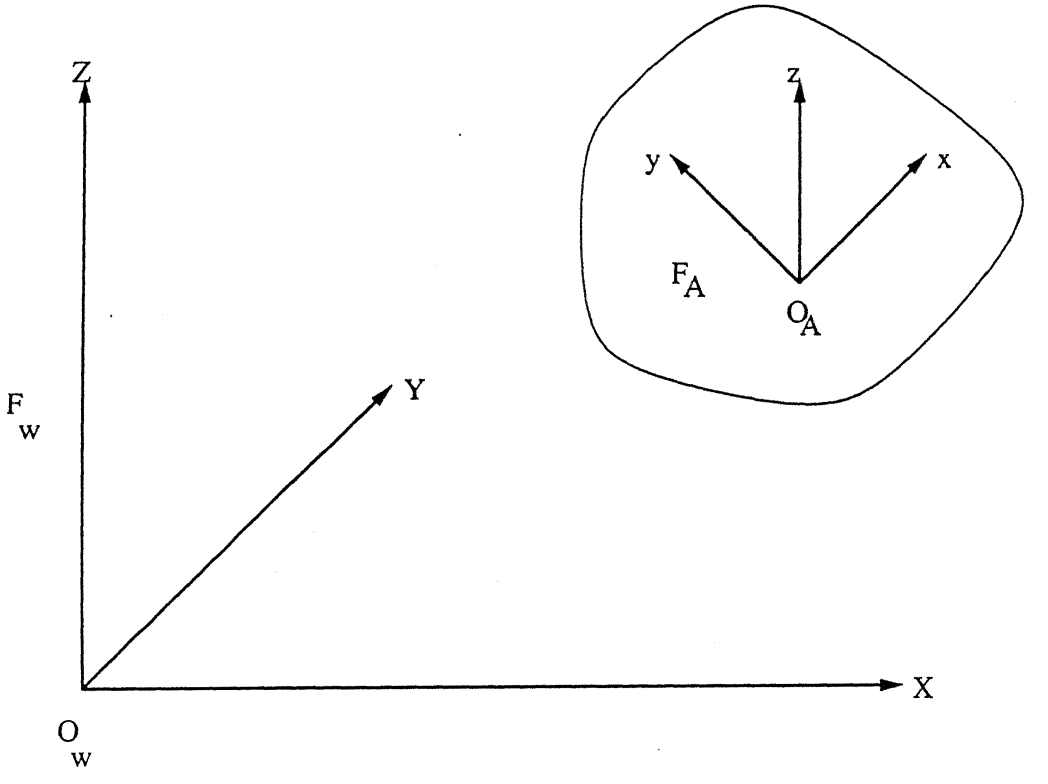


Figure A.1: The robot moves in a workspace $\mathcal{W} = R^N$, $(N = 3)$. $\mathcal{A}$ is modeled as a compact subset of R^N. A fixed Cartesian frame $\mathcal{F}_\mathcal{W}$ is embedded in $\mathcal{A}$. The configuration of $\mathcal{A}$ specifies the position and orientation $\mathcal{F}_\mathcal{A}$ with respect to $\mathcal{F}_\mathcal{W}$.

A.3 Obstacles

Let $\mathcal{W}$ contain fixed obstacles $\mathcal{B}_i = 1, 2, \ldots, q$ which we represent as closed but not necessarily bounded [13] regions of R^N. The $\mathcal{B}_i$'s denote both the physical obstacles and the subsets of R^N that represent them.

A.4 C-obstacles

Definition: The obstacle $\mathcal{B}_i$ in $\mathcal{W}$ maps in $\mathcal{C}$ to region

$$\mathcal{CB}_i = \{\mathbf{q} \in \mathcal{C}/\mathcal{A}(\mathbf{q}) \cap \mathcal{B}_i \neq \phi\}$$

$\mathcal{CB}_i$ is called **C-obstacle**. The union of all C-obstacles

$$\bigcup_{i=1}^{q} \mathcal{CB}_i$$

is called the C-obstacle region and the set:

$$\mathcal{C}_{free} = \mathcal{C} \bigcup_{i=1}^{q} \mathcal{CB}_i = \left\{ q \in \mathcal{C}/\mathcal{A}(\mathbf{q}) \cap \left(\bigcup_{i=1}^{q} \right) \neq \phi \right\}$$

is called the **free space**. Any configuration in $\mathcal{C}_{free}$ is called the **free configuration**. A **free path** between two free configurations $\mathbf{q_{init}}$ and $\mathbf{q_{goal}}$ is a continuous map $\tau : [0, 1] \rightarrow \mathcal{C}_{free}$, with $\tau(0) = \mathbf{q_{init}}$ and $\tau(1) = \mathbf{q_{goal}}$. Two configurations belong to the same connected component of $\mathcal{C}_{free}$ if and only if they are connected by a free path.

Appendix B

Transformations

B.1 Roll, Pitch, Yaw Angles

One method of describing the orientation of a frame $\{B\}$ is as follows:

> Start with the frame coincident with a known reference frame $\{A\}$. First rotate $\{B\}$ about $\hat{X}_A$ by an angle γ, then rotate about $\hat{Y}_A$ by an angle β, and then rotate about $\hat{Z}_A$ by an angle α.

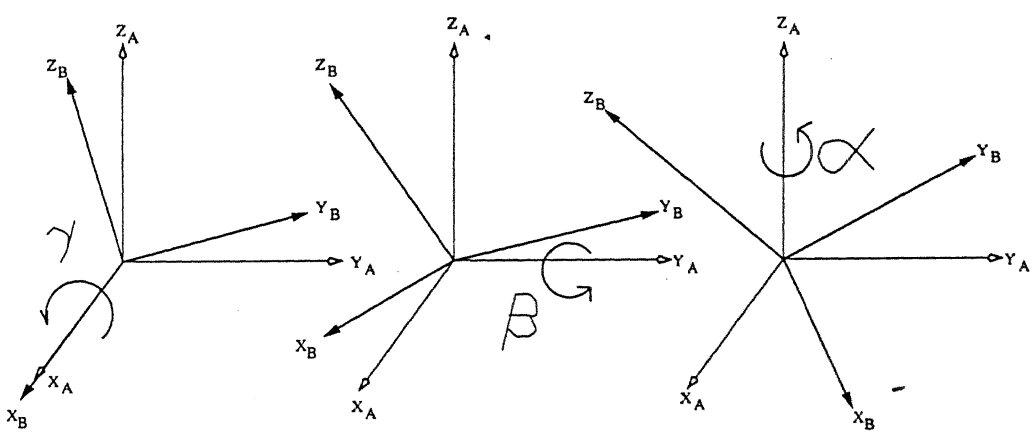


Figure B.1: **RPY angles**

Refer Fig B.1. Each of the three rotations takes place about an axis in the fixed reference frame, $\{A\}$. We call this convention for specifying an

orientation **roll, pitch, yaw** angles. Sometimes this convention is referred to as **X-Y-Z fixed angles.** The word "fixed" refers to the fact that the succesive rotations are specified about the fixed (i.e. non-moving) reference frame:

$$
\begin{aligned}
{}^{A}_{B}R_{XYZ}(\gamma,\beta,\alpha) &= R_Z(\alpha)\ \ R_Y(\beta)\ \ R_X(\gamma) \\
&= \begin{bmatrix} c\alpha & -s\alpha & 0 \\ s\alpha & c\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} c\beta & 0 & s\beta \\ 0 & 1 & 0 \\ -s\beta & 0 & c\beta \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & c\gamma & -s\gamma \\ 0 & s\gamma & c\gamma \end{bmatrix}
\end{aligned}
$$

Multiplying, we obtain

$$
{}^{A}_{B}R_{XYZ}(\gamma,\beta,\alpha) = \begin{bmatrix} c\alpha c\beta & c\alpha s\beta s\gamma - s\alpha c\gamma & c\alpha s\beta c\gamma + s\alpha s\gamma \\ s\alpha c\beta & s\alpha s\beta s\gamma + c\alpha c\gamma & s\alpha s\beta c\gamma - c\alpha s\gamma \\ -s\beta & c\beta s\gamma & c\beta c\gamma \end{bmatrix}
$$

B.2 Denavit-Hartenberg Parameters

Any robot can be described kinematically by giving the values of four quantities for each link. These four quantities are link twist (α_{i-1}), link length (a_{i-1}), link offset (d_i), and joint angle (θ_i). Two describe the link itself, and two describe the link's connection to the neighbouring link. In the usual case of a revolute joint, θ_i is called the **joint variable,** and the other three quantities would be fixed **link parameters.** For prismatic joints d_i is the joint variable and the other three quantities are fixed link parameters. The definition of mechanisms by means of these quantities is a convention usually called the **Denavit-Hartenberg notation** [12].

B.3 Link Transformations

We wish to determine the transformation which defines frame $\{i\}$ relative to the frame $\{i-1\}$. For any given robot, this transformaton is a function of only one variable. By assigning a frame for each link we have broken the problem into n subproblems (for n-linked robot) [12]. In order to solve each of these subproblems, namely ${}^{i-1}_{i}T$, we will further break the problem into four sub-subproblems. we begin by defining three intermediate frames for each link namely: $\{P\}$, $\{Q\}$, and $\{R\}$.

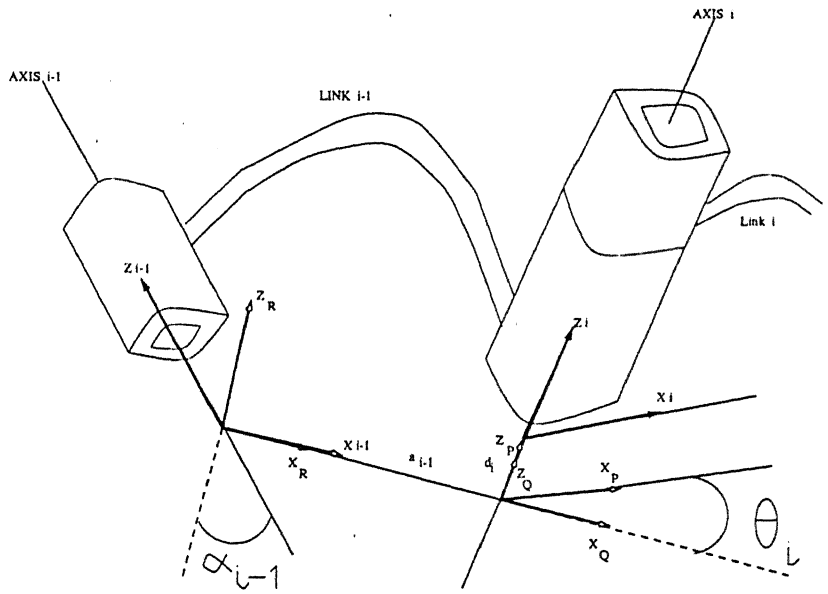


Figure B.2: **Link Transformations**

Fig B.2 shows a pair of joints with frames $\{P\}$, $\{Q\}$ and $\{R\}$ defined. Frame $\{R\}$ differs from frame $\{i-1\}$ only by a rotation of α_{i-1}. Frame $\{Q\}$ differs from $\{R\}$ by a translation a_{i-1}. Frame $\{P\}$ differs from $\{Q\}$ by a rotation θ_i, and frame $\{i\}$ differs from $\{P\}$ by a translation d_i. If we wish to write the transformation which transforms vectors defined in $\{i\}$ (^iP) to their description in $\{i-1\}$ (^{i-1}P) we may write,

$$^{i-1}P = {}^{i-1}_{R}T\ {}^{R}_{Q}T\ {}^{Q}_{P}T\ {}^{P}_{i}T\ {}^{i}P \tag{B.1}$$

or

$$^{i-1}P = {}^{i-1}_{i}T\ {}^{i}P, \tag{B.2}$$

where

$${}^{i-1}_{i}T = {}^{i-1}_{R}T\ {}^{R}_{Q}T\ {}^{Q}_{P}T\ {}^{P}_{i}T. \tag{B.3}$$

Considering each of these transformations, we see that B.3 may be written as:

$${}^{i-1}_{i}T = R_{X_{i-1}}(\alpha_{i-1})D_{X_{i-1}}(a_{i-1})R_{Z_i}(\theta_i)D_{Z_i}(d_i), \tag{B.4}$$

ultiplying the factors in B.4 we obtain the general form of ${}^{i-1}_{i}T$ as

$$
{}^{i-1}_{i}T = \begin{bmatrix}
c\theta_i & -s\theta_i & 0 & a_{i-1} \\
s\theta_i c\alpha_{i-1} & c\theta_i c\alpha_{i-1} & -s\alpha_{i-1} & -s\alpha_{i-1} d_i \\
s\theta_i s\alpha_{i-1} & c\theta_i s\alpha_{i-1} & c\alpha_{i-1} & c\alpha_{i-1} d_i \\
0 & 0 & 0 & 1
\end{bmatrix}
$$

Appendix C

Input Format

This is a sample input format for the program to map obstacles from Cartesian space to configuration space.

Statements in **bold** appear as prompts during input, while the normal font is used to show the user input. Statements in *italics* are comments for the understanding of the reader.

{0} frame is the fixed base frame of the manipulator.

Input the room dimensions <length, height, breadth> in cm: 100 100 100

Input the resolution factor :2

This factor determines the number of cells the workcell is to be divided in. It is multiplied with the value of each dimension to get the number of cells. A value of this factor more than 1 increases the resolution. For the above input the number of cells are :

$100 \times 2 = 200$

$100 \times 2 = 200$

$100 \times 2 = 200$

Input number of obstacles (max 10) : 1

The obstacles are made up of two kinds of primitives : parallelopiped and

cylindrical.

Consider yourself to be standing in front of the rectangular room. The universal frame is fixed to the far-bottom-left corner. Y-axis is from bottom to top, Z-axis points right at you and X-axis is from your left to right.

For obstacle 1
Give Position.
Co-ordinates of the origin of its local frame wrt the universal frame
: 5 5 5
Give Orientation (RPY angles wrt universal frame).
Rotation about X-axis : 0
Rotation about Y-axis : 0
Rotation about Z-axis : 0
Number of primitives : 2

Enter data for primitive 1
Press 'p' for parallelopiped or 'c' for cylindrical : p
Give Dimensions. (In its local frame defined by you.)
Length (along its X-axis) : 40
Breadth (along its Z-axis) : 20
Height (along its Y-axis) : 30
Give Position.
Co-ordinates of the origin of its local frame wrt the obstacle frame
: 5 5 5
Give Orientation (RPY angles wrt obstacle frame).
Rotation about X-axis : 0
Rotation about Y-axis : 0
Rotation about Z-axis : 0
If given obstacle intersects with the boundary at the given position and orientation, the program stops.
Enter data for primitive 2
Press 'p' for parallelopiped or 'c' for cylindrical : c

A cylindrical obstacle can be from a full solid cylinder to a sector of a hollow cylinder.

Angles are measured about the local Y-axis, with theta = 0 along the Z-axis.

Give Dimensions.
Inner radius : 5
Outer radius : 10
Start angle (between 0 and 360): 10
End angle (between 0 and 360): 270
Length (along Y-axis): 20
Give Position.
Co-ordinates of the origin of its local frame wrt the obstacle frame : 10 20 10
Give Orientation (RPY angles wrt obstacle frame).
Rotation about X-axis : 0
Rotation about Y-axis : 0
Rotation about Z-axis : 0

Input the number of links of the manipulator (max 30) : 3

If link is not along any of the axes in its local frame, input a set of points giving the mid-line curve of the link.

Input the D-H parameters for link 1 :
Give file name containing set of points or press "go" to continue :
go
Revolute/Prismatic (R/P) : r
Length : 20
alpha[0] : 0
a[0] : 0
d[1] : 0
Limits on theta[1] : 45 270

For a non-interactive execution, the user inputs can be included in the correct order in a file and submitted to the program. For the above example, the input file will be as given below.

```
100 100 100
2
1
5 5 5
0
0
0
2
p
40
20
30
5 5 5
0
0
0
c
5
10
10
270
20
10 20 10
0
0
0
3
go
r
```

20
0
0
0
45 270
bot
link2
r
0
20
0
90 200
go
p
25
0
5
0
10 5
top
x
25 10 20
90
0
0
5
0.997
20

Appendix D

Program Details

Functions used for getting input data:

- **get_room** : To get room dimensions.
- **get_obs** : To get data regarding obstacles.
- **get_prims** : To get data regarding primitives.
- **get_manip** : To get data regarding the manipulator.
- **get_base** : To get position and location of manipulator base.

All the above functions are called in **main**.

Functions used for flagging cells:

- **flag_boundary** : To flag cells belonging to workcell boundary.
- **flag_obs** : To flag cells belonging to obstacles.
- **flag_empty** : To flag cells belonging to free space.

All the above functions are called in **main**

Functions for performing the mapping

- **recursive** : To generate configurations, compute intersections, identify different C-obstacles and to store them. It is called in **main**.

- **matrices** : To calculate different transformation matrices. It is called in **recursive.**

- **intersection** : To compute intersections, identify separate C-obstacles and store them. It is called in **recursive.**

- **search** : To search in C-space for identifying separate C-obstacles. It is called in **intersection.**

- **check** : To check for neighbors in C-space. It is called in **search.**

Other functions

- **matmult41** : To multiply transformation matrices with point vectors. It is called in **main, flag_obs** and **intersection.**

- **matmult44** : To multiply two 4×4 matrices. It is called in **matrices.**

- **makemat** : To construct a 4×4 transformation matrix. It is called in **flag_obs.**

- **rewind** : To clear the workspace of cells flagged as belonging to the manipulator after each configuration. It is called in **recursive.**

- **prn_list** : To print C-obstacle points from linked list. It is called in **intersection.**

Bibliography

[1] T Lozano Peréz. *Spatial Planning: A Configuration Space Approach.* IEEE Transactions on Computers, 32(2) : 108-120, 1983.

[2] R A Brooks and T Lozano Peréz. *A Subdivision Algorithm in Configuration Space for Findpath with Rotation.* IEEE Transactions on Systems, Man and Cybernetics, SMC-15(2):224-233, 1985.

[3] O Khatib. *Real Time Obstacle Avoidance for Manipulators and Mobile Robots.* IJRR, Vol. 5, No. 1 pp 90-98, 1986.

[4] J baraquand, B Langloids and J C Latombe. *Numerical Potential Field Techniques for Robot Path Planning.* IEEE Transactions on Systems, Man and Cybernetics, Vol. 22 No. 2 pp 224-241, 1992.

[5] S Ma and E Konno. *An Obstacle Avoidance Scheme for Planar Hyper-Redundant Manipulators.* JRSJ Vol. 15, No. 7, pp. 59-64.

[6] L Kavraki and J C Latombe. *Randomized Preprocessing of Configuration Space for Fast Path Planning.* Proc. of the IEEE Intl. Conf. on Robotics and Automation, pp 2138-2145, 1994.

[7] J Lengyel, M Reichert, B R Donald and D P Greenberg. *Real-Time Robot Motion Planning Using Rasterizing Computer Graphics Hardware.* Computer Graphics, Vol. 24, No. 4, pp 327-335, 1990.

[8] J Ilari and C Torras. *2D Path Planning : A Configuration Space Heuristic Approach.* IJRR Vol. 9, No. 1, pp 75-91, Feb 1990.

[9] C Bajaj and M S Kim. *Generation of Configuration Space Obstacles: Moving Algebraic Surfaces.* IJRR Vol. 9, No. 1, pp 92-112, Feb 1990.

[10] I Gravagne and I D Walker. *Kinematics for Constrained Continuum Robots Using Wavelet Decomposition.* Robotics 2000, Proc. of the 4th Intl. Conf. and Exposition/Demonstration on Robotics for Challenging Situations and Environment, pp 292-298, Feb 2000.

[11] Y Xu and M C Nechyba. *Fuzzy Inverse Kinematics Mapping: Rule Generation, Efficience and Implementation* tech. report, CMU-RI-TR -93-02, Robotics Institute, Carnegie Melon University, Jan 1993.

[12] J J Craig. *Introduction to Robotics* Addison-Wesley Publishing Company Inc. 1986

[13] J C Latombe. *Robot Motion Planning* Kluwer Academic Publishers, USA., 1991

[14] D E Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley Publishing Company, Inc. 1989

[15] D E Goldberg and J Richardson. *Genetic Algorithms with Sharing for Multimodal Function Optimization.* Genetic Algorithms and their Applications: Proc of the Second Intl. Conf. on Genetic Algorithms, pp 41-49.

[16] K Deb. *Multi-Objective Evolutionary Algorithms: Introducing Bias Among Pareto-Optimal Solutions* KanGAL Report No. 99002, Kanpur Genetic Algorithms Laboratory (KanGAL), Deptt of Mech Engg., IIT Kanpur, India.